2013

A model for systematically investigating relationships between variables that affect the performance of novice programmers

Vivian Campbell
Edith Cowan University

A model for systematically investigating relationships between variables that affect the performance of novice programmers

Vivian Campbell
Edith Cowan University, v.campbell@iinet.net.au

Edith Cowan University

Copyright Warning

A model for systematically investigating relationships between variables that affect the performance of novice programmers.

Vivian Campbell
MSc

This thesis is presented in fulfilment of the requirements for the degree of Doctor of Philosophy

Faculty of Health, Engineering and Science
EDITH COWAN UNIVERSITY

24th October 2013

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

ABSTRACT

This research was motivated by an interest in novices learning to program and a desire to understand the factors that affect their learning. The traditional approach to performing such an investigation has been to select factors which may be important and then perform statistical tests on a few potential relationships. A new research model is proposed and tested to ensure that a thorough and systematic investigation of the data is performed. This thesis describes the data, defines the model and explains the application and validation of the model.

The research process is managed by a control algorithm that is the heart of the model. This algorithm is seeded by a hypothesis that connects two variables of interest and dictates the testing of a series of hypotheses; as it does this, it also delves deeper into the data to identify additional relationships.

In this research the model was applied to investigate the relationships between: learning style and achievement; programming behaviour and achievement; and learning style and programming behaviour. Learning style was assessed using Kolb's Learning Style Inventory, achievement was based on exam score and programming behaviour was extracted from a log of student activities using a programming tool. The largest number of significant relationships was found between aspects of behaviour and achievement.

The model was validated by classifying the significant hypotheses based on the research model's tree structure, the section of the programming tool in use and the literature. These three classification schemes provided a structure to explore their similarities and differences. The model was thus demonstrated to be robust and repeatable by comparing the results with those from both using a programming tool, and expert opinion.

This research has revealed several attributes of the learning behaviour that affected the students' results within this group, including aspects of timeliness and overall volume of activity. These are suitable targets for future investigations.

The research model could be applied to other data sets where an in-depth investigation into pairwise data is required.

DECLARATION

I certify that this thesis does not, to the best of my knowledge and belief:

(i) incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education.

(ii) contain any material previously published or written by another person except where due reference is made in the text; or

(iii) contain any defamatory material.

I also grant permission for the Library at Edith Cowan University to make duplicate copies of my thesis as required.

Signature: …….

Date: ……..

ACKNOWLEDGEMENTS

I am grateful to my supervisors, Mike Johnstone and Geoffrey Roy, for their advice, assistance and patience. Also, to Peng Lam, who contributed significantly to the direction of this work.

I am indebted to Lynne Fowler and Jocelyn Armerego, for their encouragement in those early days.

Many thanks to colleagues and friends, Lynelle Watts and Julie Goyder, for their general support and words of reassurance.

To my family and friends who have kept asking, “Is it finished yet?”, I am relieved to finally be able to say, “YES”.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

This thesis was originally going to be an investigation of the processes used by novices when learning to program. However as the research progressed and I became more familiar with the literature, it became increasingly apparent that there was often a lack of rigour in the research processes that are used in this field. This was especially true of the seemingly unplanned nature of the process of selecting factors which might influence novice programmers' achievement. My increased awareness of the need for a more structured method, led to the research model becoming central to this thesis.

This research was motivated by an interest in novices learning to program and a desire to understand how their behaviour affected their learning. The objective was to create a research model that would facilitate a systematic and thorough investigation of relationships in the available data to characterise the learning situation. This thesis tracks the development, application and validation of such a model.

The most important component of this research model is an algorithm that directs the process of exploring the data through repeated hypothesis testing. The model was trialled by application to the domain of learning programming, wherein, it facilitated the examination of the effectiveness of the behaviours of students as they used a software tool in university computer laboratories. The relationships between student behaviour, preferred learning style and achievement were comprehensively explored to discover any significant connections between these variables.

This chapter positions the study, justifies the selection of this topic, clarifies the questions for investigation, explains the research contribution and finally outlines the rest of the thesis.

1.1 Background to the study

This research is associated with the field of Computer Science Education Research (CSER). An area that was, perhaps optimistically, announced in 2002 as "having arrived mainstream" (Dale, 2002) and shortly afterwards in

2004, identified as still being an "emergent" discipline (Fincher & Petre, 2004). CSER is dependent upon educational theory, however, it also requires a thorough appreciation of the complexity of the processes of computing (Fincher & Petre, 2004). The discipline has been further consolidated by developments that include the International Conference on Computing Education Research (ICER) that has been held annually since 2005.

It should be noted that the name of the field or fields is not consistent; Computing in ICER and Computer Science in CSER. There is also substantial overlap with the disciplines of Software Engineering, Information Technology and Information Systems.

While Comer et al. (1989) believe that there is a long held view that *Computer Science equals Programming*, there is a more general acceptance that programming is just one of several core skills and knowledge areas required in a tertiary Computer Science (CS) course. Indeed programming has been an essential component of the CS curriculum since first developed by the Association for Computer Machinery (ACM) in 1968 (Atchison, Conte, Hamblen, Hull, Keenan, Kehl, McCluskey, Navarro, Rheinboldt, Schweppe, Viavant, & Young, 1968). Today various aspects of programming are at the heart of the CS discipline, which spans from systems infrastructure, through software methods and technologies to application technologies (Shackelford, McGettrick, Sloan, Topi, Davies, Kamali, Cross, Impagliazzo, LeBlanc, & Lunt, 2006). In Computing Curricula 2005, the ACM/IEEE Joint Task Force pointed to the great diversity of computing programs that were continuing to develop in the 21st Century; however they also stated that there continued to be one common element in all of these courses and that this was programming (Shackelford et al., 2006).

While a wide range of views on when a focus on programming should occur within the curriculum have been expressed, there are several reasons that programming is often placed early as discussed by Roberts et al. (2001). These include:

- Programming is required by advanced CS courses, i.e. it is often considered to be a foundational element

- Demand from other courses outside CS, i.e. CS departments provide "service courses"
- Students like programming, i.e. writing programs is seen to be more interesting than learning the underlying principles.

So programming remains an important component of the CS curriculum and the process of learning and teaching programming was acknowledged as one of the grand challenges in computing (McGettrick, Boyle, Ibbett, Lloyd, Lovegrove, & Mander, 2005). The area of research is therefore rich in opportunities.

1.2 Rationale for the study

Many novice programming students find the process of learning to program extremely difficult (Mayer, 1981; Hoc, 1990; McCracken, Wilusz, Alstrum, Diaz, Guzdial, Hagan, Kolikant, Laxer, Thomas, & Utting, 2001; Robins, Rountree, & Rountree, 2003; Lister, Adams, Fitzgerald, Fone, Hamer, Lindholm, McCartney, Mostrom, Sanders, Seppala, Simon, & Thomas, 2004; McGettrick et al., 2005; Greyling, Cilliers, & Calitz, 2006; Norris, Barry, Fenwick, Reid, & Rountree, 2008). Despite decades of research it is still not clear which teaching and learning pedagogies will best promote the development of the required skills in a way that is attractive to students (Weinberg, 1971; Mayer, 1981; Hoc, 1990; Winslow, 1996; Felleisen, Findler, Flatt, & Krishnamurthi, 1998; McGettrick et al., 2005; Gomes & Mendes, 2010). While many students are successful in an introductory programming course at a tertiary level, others find it a major obstacle. There have been numerous attempts to identify the attributes of students who will become successful programmers but these have not been very productive (Allert, 2004; Bennedsen & Caspersen, 2005; Bergin & Reilly, 2005). This study attempts to explore these issues through a formal analysis of the recorded behaviour of students in the early phase of their learning to program. Other studies have used similar techniques (Jadud, 2005b; Spacco, Strecker, Hovemeyer, & Pugh, 2005; Norris et al., 2008) but it will be demonstrated that the data analysis in this study is more systematic and comprehensive.

This research is an empirical study of the actions of novice programmers. It provides a research model to investigate student behaviour and identify whether any activities of the student are particularly effective in improving learning outcomes. This could then inform the instructional processes to better support these useful activities.

An introduction to programming is often placed early in CS courses, and it may be that the problems faced by students in programming are compounded because they are at the same time struggling to establish study habits that are needed for success. The regularity of study will be one aspect that will be investigated in this research.

1.3 Research questions

This thesis aims to answer the question, “Can a model for systematically investigating relationships between variables that affect the performance of novice programmers be constructed? “

It will do this through seeking an answer to the following research question:

RQ1. Can a model be constructed to systematically test relationships between learning style, behaviour and achievement?

An answer to this primary question will be sought in part by using three secondary research questions:

RQ2. Does learning styles affect achievement?
RQ3. Does student behaviour affect achievement?
RQ4. Does learning styles affect behaviour?

An attempt will be made to answer all these questions through the development and application of a research model. The data to be used was collected by a monitoring program that recorded a log of the activities of the students whilst they undertook their programming exercises in a university laboratory.

The various theories of student learning styles suggest that there are many ways of learning. The learning styles of the students were assessed using a standardised testing procedure, and this information was used to correlate or categorise students against both their observed learning behaviour and their achievement, as measured by end of semester examination results.

1.4 Research contributions

This research aimed to construct a research model to systematically test relationships. The model was applied to a set of three related datasets:

- A substantial record of student activities (a log) as they undertook their programming exercises.
- The measured learning outcomes of the grades achieved at the end of semester examination and assessed project work.
- A classification of students into their leaning styles as determined by a standardised measurement (the Kolb Learning Style Inventory).

These datasets offer the opportunity to design and test hypotheses that relate various dependent and independent variables in an attempt to identify interesting or useful relationships. It will be through the identification of interesting hypotheses that conclusions may be drawn about which learning activities appear to be the most useful in improving outcomes. To achieve this, a research model has been developed.

This model provides a structure that allows for the detailed investigation of pairwise attributes of an object. In the application of the model, the object of interest is the student, while the attributes are student behaviour, achievement and learning style. The student's behaviour provides an insight into the activities with which the student engages that may affect his/her learning. By recording this behaviour at an appropriate level of detail, a data set is collected which may be used for empirical hypothesis testing. This research was facilitated by the use of P-Coder, an Integrated Development Environment (IDE) which recorded an event log file (see sections 3.5 and 3.6).

The definition of the model will present the means by which the data can be analysed systematically and provide the structure to answer the primary research question 1; whilst the secondary research questions 2 – 4 will be answered by three separate applications of the model. A single application requires the selection of the independent and dependent variables in order to answer a specific question (section 4.7). Each application of the model results in the formation of a hierarchy of hypotheses and progresses the investigation from more general aspects of the data to the more specific aspects by drilling down into the data.

The research questions were selected to find which aspects of students' behaviour or learning style influenced their learning, the intention being to attempt to identify which might be effective behaviours, so that in the future these could be encouraged through modification to the teaching and learning process.

In summary the main contribution of this thesis is the development of a research model that provides for a systematic empirical investigation into the data. The model was trialled by its application to the domain of novice programming in which relationships between aspects of student behaviour, learning style and achievement were examined.

1.5 Overview of the thesis

The thesis describes a research model that was devised to guide the thorough investigation of related data sets. It was trialled by application to the activities of students as they learn to program. The results obtained from the application of the research model are discussed relative to alternatives.

Chapter 2 positions this research in the literature. It begins by comparing several attempts to classify the CSER literature and then turns to explore the components that are required to build a systematic model. The domain of introductory programming is explored and an examination of the research into factors that might affect success is included. A brief consideration of tools for novice programming, followed by the relevance of learning styles inventories concludes this chapter.

Chapter 3 introduces the course, (G108 Engineering Computing), the students and P-Coder, the programming tool that provides a mechanism for the behavioural data collection. The process of collecting the data and the nature of the data are explained and finally the learning styles inventory used in this thesis is justified.

Chapter 4 explains the selection of the research approach, introduces the research model and explains its structure. It also explores each of the model's components to demonstrate their configuration and associations.

Chapter 5 is the manifestation of this model in the domain of novice programmers, specifically when applied to the subjects described in Chapter 3. The application of the research model is explained and the analysis process is examined through detailed investigation into several cases that describe decision points. The complete hypothesis trees are presented and the chapter concludes with a discussion of the operational issues that were uncovered.

The analysis and discussion of the research in Chapter 6 is centred on three classification schemes that are used to explore the results described in Chapter 5. The purpose of the schemes is to validate the model.

Chapter 7 concludes the thesis. It begins with a summary of the major findings and reflections on the research. Next, suggestions for future research are discussed and finally speculations are offered on aspects of learning and teaching that could be considered based on the results.

CHAPTER 2 LITERATURE REVIEW

In Chapter 1 the rationale and aims of this study were explained; this chapter establishes the research context. Section 2.1 endeavours to understand the nature of the computing education research and uncovers the fact that several classifications of computing education literature, have concluded that there are comparatively few quantitative studies. Section 2.2 explores investigations into the data, with the objective of discovering some concepts that might be useful in building a systematic model. This includes studying techniques for hypothesis development and testing which may become part of the research model.

Section 2.3 notes that programming, which has an important place in the computing curriculum, continues to be an impediment to the progress of many students. There have been a number of studies on the factors affecting the success of novice programmers; the findings of these are reviewed.

Many different tools have been proposed to assist novices to develop the required skills to become successful programmers. Since the early paper based tools, IDEs have been developed to ensure that novices are not overwhelmed by complex commercial products. Some of these purpose built tools have also been adapted to provide for observational studies of novices. Section 2.4 studies the tools and their use in examining achievement. Finally, many authors have considered learning styles to be an important factor in student learning and in order to answer the research questions an examination of this field is essential.

2.1 Computing education research

This section examines classifications of the computing education literature and concludes that whilst there has been a recent increase in empirical research, quantitative studies remain in the minority. Many authors have attempted to classify the literature in computer science education research (Fincher & Petre, 2004; Valentine, 2004; Pears, Seidman, Eney, Kinnumem, & Malmi, 2005; Simon, 2007; Simon, Carbone, de Raadt, Lister, Hamilton,

& Sheard, 2008; Joy, Sinclair, Sun, Sitthiworachart, & López-González, 2009); each of these has taken a different approach and had slightly different objectives.

Fincher and Petre (2004, p. 3) aimed simply to "identify broad areas that motivate researchers", while Joy et al. (2009) sought to develop a taxonomy that would assist new researchers to know where papers appear. Valentine restricted his observations to only first year core Computer Science courses where he aimed to "discover the state of the art" and look at the development of the discipline. Simon's goal was to investigate the computing education research in Australia and New Zealand with particular interest in identifying the number of research and practice papers (Simon, 2007). There has been no general consensus within these classification systems, perhaps because of these different objectives.

Valentine's (2004) meta-analysis of papers presented to the SIGCSE Technical Symposium on issues related to the introductory sequence of a computer science major resulted in a classification system with only six categories. Valentine compared the CSER papers published in the decade 1994-2003 with those published in the previous decade 1984-1993. He would have liked to see an increase in the number of experimental papers but found instead that this was relatively stable over the period analysed.

Pears et al. (2007) are critical of Valentine's categories for not being mutually exclusive; in fact Valentine himself described his experimental category as being "pre-emptive". That is, if a paper could be classified as both experimental and another category then, it was classified experimental.

Fincher and Petre (2004) identified ten broad areas that were not mutually exclusive. These ten were aggregated into only three by Pears et al. (2005), who also added a fourth classification.

Simon's system for classifying papers (Simon, 2007) was distinctive and more complex in that it identified four dimensions. Simon stated that classification of a paper on one dimension was independent of the others. Simon supported Valentine's earlier finding by reporting that whilst there had been an increase in empirical research, quantitative studies remained in the minority (Simon et al., 2008).

2.2 Building a systematic model

This section will study the components required to build a systematic model. The emphasis is on locating potential constituent parts for an empirical exploration of the data. This section has three parts; first empirical research in computer science education is explored, then methods that have been used to study observational data are uncovered and finally methods of generating new knowledge through hypothesis formation and testing are investigated.

2.2.1 Empirical research

A survey of approaches to teaching programming by Lemnos (1979) concluded that the lack of empirical research meant that methods of teaching and learning could not be effectively compared. This was still an issue 25 years later, according to Fincher and Petre (2004). This opinion was supported by Joy et al. (2009) who stated that "practice-based, technology-driven reports" remain the prevailing classification in this field of study; more recent examples include (Guo, 2013) and (Black, Bruce, Homer, Noble, Ruskin, & Yannow, 2013). There are some niches, such as the International Conference on Computing Education Research (ICER), where more empirical research can be found (Simon et al., 2008).

In research that involves people, the strict requirements for an experiment may not be practicable and treating different groups of students in a dissimilar manner may be ethically indefensible. For example, if a new teaching technique is believed to be superior then it could be considered unfair to withhold this technique from some students to create a control group. Counter to this argument is the need to research new techniques and tools in an empirical way. It is enlightening to compare the trivial, 'Robots are fun', conclusion of Weber Becker (2001, p. 53) and with the statistical results reported by Cilliers, Calitz and Greyling (2005).

There remain numerous practical issues in experimental research in education because so many variables are involved in the complex processes of learning and teaching. These constraints mean that it is often not possible to use all of the rigour strictly required by the scientific method, however,

there is still a need for more empirical studies in CSE (Fincher & Petre, 2004; Simon et al., 2008). Regardless, some empirical research has explored the factors which influence novice programming success.

Comparing methods or techniques of teaching is fraught with problems (D. Cook, 1967; Fraenkel & Wallen, 2006). Any new technique that is being studied may result in an improvement simply because of the altered behaviour of those being placed under the spotlight. The name given to this phenomenon is The Hawthorne Effect[1]. Bellamy (1994, p. 244), when investigating the use of pseudo-code by programmers, stated that, "to understand exactly how the different tools support the programming task requires empirical studies comparing different languages, different environments and different programming tasks".

Despite relevant groups and workshops being active for several years, including the Empirical Study of Programming and the Psychology of Programming Interest Group (PPIG), the lack of empirical research has been evident. Randolph (2007), based on papers published between 2000 and 2005, noted the lack of experimental design in Computer Science Education (CSE). Randolph suggested that anecdotal evidence is insufficient for hypothesis testing and urged CSE researchers to undertake research that will systematically test hypotheses so that research does not generate a literature of 'folk conclusions' which are in danger of becoming accepted, despite being unproven. Tenenberg et al. (2008, p. 215) stated that papers being submitted for review often lacked "evidence for the learning claims" made.

However, a few researchers have used empirical methods. Ko, Aung and Myers (2005) used screen-capture to study the percentage of time that experienced programmers spent on maintenance tasks with the objective of finding ways to improve professional IDEs. A similar objective was behind a study where screen-capture was used again but on novice programmers (Hundhausen, Brown, Farley, & Skarpas, 2006). Phenomenology has been used to investigate the processes of learning in CS (Bergland & Wiggberg, 2006).

[1] Although this term is widely used, the validity of the original Hawthorne study has been questioned (Clark, 1999).

The BRACElet project sought an understanding of student learning through the empirical evidence collected largely from the answers to exam questions (Lister et al., 2004; Venables, Tan, & Lister, 2009; Lister, Clear, Simon, Bouvier, Carter, Eckerdal, Jackov, Lopez, McCartney, Robbins, Sepp, & Thompson, 2010). The motivation behind BRACElet has some similarities to this research but the method is quite different. This research seeks an effective method to identify particular behaviours that lead to success.

2.2.2 Using observational data

Observational studies are those that are based on recording and analysing data. They are sometimes used when controlled experiments are impractical or unethical (Benson & Hartz, 2000). This section uses the term observational to define studies where the data are recorded by observation of the subjects' behaviour or actions. It may be a manual recording of an action or activity, a manually coded translation from a video or computer recording, or an entirely automated computer log file that records the subjects' computer use.

Sequential Data Analysis (SDA) has been used to look for meaning in such data. This is of interest to this study because the record of student behaviour, that is accessible, is a sequential log of student activities whilst using a specific software tool. There are several branches of SDA in computational science covering diverse applications, including DNA sequencing, linguistics and music, as well as computer science (Sankoff & Kruskal, 1983). The most relevant forms of SDA for this study involve log files, also called audit trails, for the automatic recording of discrete user interactions with a computer. The relevance of this type of research to CSER was noted by Lister (2010, p. 23) who stated that routinely logging data was a positive move that could assist with "systematically studying why one student passes while another fails".

These methods have been used to study Human Computer Interaction (HCI) (Guzdial, 1993), to discover software engineering process models (J. E. Cook & Wolf, 1995) and to explore the sequences used in multimedia tools (Judd & Kennedy, 2004). Their benefits and disadvantages include that data

can be easily and unobtrusively collected and the difficulty of analysing vast quantities of data, respectively.

The granularity of the data that is recorded should be suitable for the type of analysis that is intended, for example, Bergadano, Gunetti and Picardi (2002) used key-stroke data as an alternative to physiological features to authenticate individuals. However, more coarsely grained data was appropriate when using SDA to study interactions with a multimedia training package (Judd & Kennedy, 2004).

User behaviour can be recorded with observer logging, voice and video recording, screen capture and a variety of automated logging techniques, ranging over many levels of granularity. Much of the relevant research sought to investigate HCI, although an interesting application that used these methods investigated the actions of air traffic controllers to ascertain the feasibility of automating the processes they used (Vortac, Edwards, & Manning, 1994). Each of the logging techniques has benefits and drawbacks depending on the type of analysis required. However, in a laboratory environment, automated logging provides an accurate record of events and an efficient means of collecting data. A drawback is that automated logging may not provide as rich evidence of events as the other methods which require human interpretation (Renaud & Gray, 2004).

An activity log was used in the Hackystat project (Johnson, Kou, Agustin, Chan, Moore, Miglani, Zhen, & Doane, 2003) which attempted to automate the Personal Software Process (PSP) for improving software engineering practices. It assisted the individual to understand the processes that they actually use. At the core of the technique are accurate records of time spent on tasks to assess productivity (Humphrey, 1995). These records can then be used to assist with estimation and eventual improvement in technique.

One of the difficulties with PSP is that software engineers have not always kept entirely accurate records, possibly because they become very engrossed in tasks. The use of Hackystat led to the discovery of several variations between the automated and manual recordings of the software engineers' activities. It was determined that using the automatic recording of process reduced the need to context switch between working and recording work

and thus permitted them to concentrate more on the task. Thus, the PSP became less onerous (Johnson et al., 2003).

In audit trail analysis the various levels of granularity pose specific and quite different issues. Keystroke monitoring produces vast amounts of data that may need to be sifted through, analysed and summarised before being useful. The GRUMP project (Gray, McLeod, Draper, Crease, & Thomas, 2004) kept extremely detailed records for later data mining, the objective being to ensure that all required data were available for research questions that were still to be formulated.

User interface events generated by the operating system or Java Abstract Windowing Toolkit may be selected in preference to keystrokes to reduce the data volume but the problem of analysis is compounded by the difficulty of identifying important details in the data (Hilbert & Redmiles, 2000; Renaud & Gray, 2004). For example, Renaud and Gray (2004) studied the tasks that users undertook following an interruption and first had to clean the data. The issues included that certain user actions that they wished to identify did not generate an event and also there was some interference from the operating system, such as when screensavers becoming active.

This review will return to the use of observational data but specifically when applied to novice programming in section 2.3.2, after some of issues of learning and teaching novices have been addressed.

2.2.3 Hypothesis formation and testing

New knowledge can be created by several means:

- a challenge to the status quo and the identification of counter examples
- by chance, followed by the acknowledgement of the importance discovery
- by forming a hypothesis as a possible explanation of an issue. Subsequently testing the hypothesis and forming the resulting conclusion.

The last of these can assist with the objective of creating a model to systematically test relationships and so is an approach that will be explored.

Gilmore (1990) identified four types of research that motivate the collection of data in empirical studies of programmers, hypothesis testing, comparisons, evaluations and exploration. However, he emphasised that there may be overlap of these categories in a single research project. The first type, hypothesis testing, is carried out within a formal theoretical framework in an experimental situation. The theoretical framework places the research in context which both connects the project to previous work and provides for a base on which the results can be explained. Statistical tests are used to assess the experimental result in a process that establishes the probability that the results occurred by chance.

It is unfortunate that Gilmore used the term hypothesis testing, to name what is more widely known as experimental research because hypothesis testing is just a part of this first research type and also often used in the second. 'Comparisons' is a form of research where the objective is to detect differences rather than define a causal effect. The meticulous detail of experimental conditions need not be met since the goal is observation rather than explanation. Gilmore states that "[Comparisons] are excellent at stimulating hypotheses and theoretical frameworks"(Gilmore, 1990) .

Gilmore's remaining types are evaluations and explorations. Evaluations are methods of searching to improve a process by looking at many measures, perhaps including "subjective" ones which are selected from the available data. Whilst an exploratory study is appropriate when investigating how tasks are completed, the data may be ill-defined and expansive and further it may be problematic to assemble; qualitative analysis is often suitable. So an exploratory study may be a preliminary study that raises several questions from which hypotheses can be formed and tested. Those found to be significant may indicate a direction for further research.

Another approach to generating hypotheses to be tested is through Data Mining. This is "the process of automatically discovering useful information in large data repositories" (Tan, Steinback, & Kumar, 2006, p. 2). It has been used to generate hypotheses to enhance the probability of the acquisition of new knowledge (Bhargava, 1999; Ping & Garcia, 2010). Association analysis can be used to discover interesting relations (Agrawal, Mannila, Srikant, Toivonen, & Verkamo, 1996). Domain knowledge can

regulate data analysis to avoid the inevitable combinatorial explosion of trial and error searches (Ping & Garcia, 2010).

Semantic networks have been added to databases so that the defined structure of database attributes can assist the process of searching for new hypotheses. This is important because it provides for new hypotheses to be formed around existing significant hypotheses using a much more constrained search for new knowledge than would otherwise be possible.

Two hypothesis generation methods, induction and analogy were proposed by Ping and Garcia (2010) . They suggested that when a relationship was discovered between data attributes then it is reasonable to suppose that any children and/or siblings may also have a connection; although the explanation given (Ping & Garcia, 2010, p. 662) actually moves in the reverse direction from child to parent.

If a relationship is uncovered of the form (*v's child* → *u*), where independent variable (*v's child*) implies dependent variable (*u*), induction is defined by forming a hypothesis with the parent of the independent variable and the dependent variable (*v* → *u*). Hypothesis generation via analogy is similar except that siblings rather than children are used.

This technique was applied to a public health dataset that contained 105 attributes and 1920 new hypotheses were generated. Although this is a large number of hypotheses to test, the potential combinatorial explosion of these attributes has been kept under control with this technique. Ping and Garcia (2010) reported the result as promising.

The methods used by Ping and Garcia to generate additional hypotheses are important for this research. It will be seen in Chapter 4 that to explore all promising relationships, the research model directs drilling down into the data with the aim of identifying hypotheses and testing the significance of any child variables.

The generation of hypotheses is a central problem when looking for internal relationships in large datasets. The challenge in this research is to formulate an approach that can identify such relationships within the scope and structure of the available data. The model, to be proposed in Chapter 4, based on the data described in Chapter 3, will provide an appropriate

strategy which has the potential to offer significant insights into any relationships between student behaviour, student learning style and performance in the area of learning to program; even if these relationships are concealed deep within the data.

2.3 Introductory programming

This section explores the domain that will provide the data that is used to apply the model. First by examining the issue of teaching novices to program and secondly by examining attempts to identify factors that predict success in programming.

2.3.1 Teaching novices to program

This section examines the nature of programming and reveals that students still have difficulties in learning to program. Despite efforts to improve teaching, many introductory programming courses still have a high failure rate. Programming continues to be a major hurdle for novices, especially since it often occurs near the beginning of tertiary computing courses.

Programming is fundamental to achieving one of the major goals of Computer Science, that is, creating a machine executable model of a real world problem (Dourish, 2001). However, programming is also a design task, as are architecture, music composition, choreography and creative writing. For computer programming this task is "piecing together a set of programming language instructions that will solve a specified problem" (Pennington & Grabowski, 1990, p. 24). This definition concentrates on the composition aspect of design but it does not capture the comprehension aspect of understanding a design.

Brooks (1983) identified that these two fundamental processes were reverse transformations between the problem domain and the domain of the program. Composition translates from the *what* of the problem domain to the *how* of the programming domain, whilst comprehension moves from the *how* to the *what*. A successful programmer must be proficient in both of these basic processes since the composition process requires

comprehension. There are two circumstances when this is most evident, when reading the code for understanding during testing, and when resuming the programming task after some time away from it.

Experienced programmers recognise that there are several stages of development in the programming process and, while educators do stress their importance, novice students continue to see the task of programming as simply coding. They continue to under-emphasise the other phases in the process of problem solving. This may be improved by directing the curriculum more directly at the problem-solving strategies that are not obvious to the novice (de Raadt, Toleman, & Watson, 2004).

Experts see a program as a collection of semantic structures (Petre, 1990) and programming is therefore selecting the appropriate pieces from these known program chunks. In contrast, beginners often perceive programming as wrestling with the compiler to create a syntactically correct program. This view is not helpful in solving problems because the emphasis on syntax, rather than the semantics of code, can ultimately result in programs that compile but still have logic errors.

Learning to program is difficult for many students, the main problem being unfamiliarity with every aspect of programming. Soloway (1986) viewed programming as having two objectives; first, constructing the mechanism that provides the instructions to the computer on how it will execute, and second, creating an explanation that will assist the human reader understand the strategy used. These can be seen as mirroring the two objectives of programming languages that is, being readable by both the computer and humans. Soloway suggested that placing a greater emphasis on human readability would foster better generic problem-solving strategies.

A multi-national study (McCracken et al., 2001) involving the ITiCSE 2001 Working Group concluded that after an initial programming course, many students could not create programs to solve problems. Several of its own inconsistencies were highlighted in this study. These included, whether the exercise was formally assessed, the appropriateness of the problem and if the language used confounded the challenge (students used C++ or Java which differ in numeric input facilities).

The cognitive domain of the Taxonomy of Education Objectives (Bloom, 1984) (Bloom's taxonomy) places reading code at a lower level than problem solving. These lower level skills were the subject of another multi-national and multi-institutional study. The BRACElet project (Lister et al., 2004; Lister et al., 2010) investigated questions and answers from exam papers and classified students level of understanding.

Also reporting from the same project, Venables, Tan and Lister (2009) concluded that students should master these essential programming skills in this order.

1. Tracing code.
2. The overall meaning of a code segment.
3. Writing code.

It has been shown that programming is difficult for novices and there is no consensus on how this situation can be improved. There have been some steps forward in describing the difficulties faced by novice programmers but a full understanding is still far from being achieved.

2.3.2 Factors affecting success

Numerous attempts have been made to predict the success of novice but few of the factors that have been selected that might affect success have been found to be significant.

Motivation is thought to be a major factor affecting performance (Biggs & Tang, 2007) however it is one that appears to be rarely studied at the tertiary level and certainly not one that has been greatly researched in Computer Science Education; although motivation was recently reported as being positively correlated with grade (Gomes, Santos, & Mendes, 2012). Both actual cost and opportunity cost are high for all tertiary students[2] so that extrinsic motivational factors are certainly present. According to Biggs and Moore (1993) motivation is a function of expectation of success and the value of achievement. This is supported by the finding that intrinsic motivation is higher for students with some experience (Carbone, Hurst, Mitchell, & Gunstone, 2009).

[2] In Australia and many other countries

Ramalingam, LaBelle and Wiedenbeck (2004) used a self-efficacy (SE) scale of programming confidence standardised by Cantwell Wilson and Shrock (2001). The SE test was given both at the start of the semester and at the end when it was accompanied by a test on program comprehension. It was not surprising that student SE score improved over the course of the semester and it was correlated, albeit at a low level, with the test; however no correlation was found between the initial SE score and test score. Interestingly this indicates that the initial confidence of students in their programming ability at the start of the semester is not correlated with their final achievement.

This result was confirmed by Ventura and Ramamurthy (2004) specifically in relation to self-efficacy. They also found no evidence that prior programming experience affected success, although they did uncover a relationship between Java experience and exam score.

In a phenomenographic study, Eckerdal and Bergland (2005) interviewed novice programming students in an attempt to uncover the students way of experiencing learning to program. They concluded that to enable success students needed to reach the understanding that:

> ".. learning to program is a way of thinking, which enables problem solving, and which is experienced as a method of thinking"

There are many other studies that have looked for factors or attributes that influence the success of students when they are learning to program (Whipkey, 1984; Hagan & Markham, 2000; Blackwell, Whitley, Good, & Petre, 2001; Byrne & Lyons, 2001; Cantwell Wilson & Shrock, 2001; Ventura, 2003; Bennedsen & Caspersen, 2005; Bergin & Reilly, 2005; Wiedenbeck, 2005; Jones & Burnett, 2007). The factors studied are predominantly of two particular types. First, attributes of the students before they begin their course; such as, high-school mathematics, programming, science and even foreign language scores, gender and preferred learning style at the start of the course. Second, attributes obtained during the study

period and including assignment scores and students' confidence and perception of their own understanding.

Byrne and Lyons (2001) investigated attributes that might predict the success of humanities students in a first course on programming and logical methods, these included learning style, gender, prior experience and previous academic performance in mathematics, science, programming and languages. An unusual aspect of this study was that 61% of participants were female (compared to an estimated 70% male in the majority of cohorts). The only factors investigated that had statistical significance were high school results in Science, Programming (low numbers) and to a lesser extent Mathematics. Some small differences in results occurred between Kolb's learning style groups but these were too small to claim significance, although they were concluded to be worthy of further investigation.

A difference in performance due to the learning style was supported by Thomas, Ratcliffe, Woodbury and Jarman (2002). They used Felder-Silverman Learning Style Model which allows classification along four axes, active/reflective, sensing/intuitive, visual/verbal, sequential/global. Both exam and course marks were available as measures of success but the exam was selected with no justification provided. The reflective learners were more successful than active learners and verbal learners were more successful than visual learners.

Goold and Rimmer (2000) supported the active/reflect difference but also considered Kolb's LSI and found a slightly larger differentiation on the Abstract Conceptualization dimension. Pillay and Jugoo (2005) found contradictory results in two studies that compared the achievement of KLSI groups, one reported Accommodators being more successful that Divergers and the other study reported no difference. They also found performance in Mathematics and a different home language to the instructional language to be a positive and negative indicator respectively.

Allert (2004) also found a very weak correlation between student results and the active-reflective scale and visual-verbal scale. Bishop-Clark and Wheeler (1994) used Myers-Briggs Type Indicator but found the only difference in performance from several tests on various dimensions was that sensors had a significantly higher score than intuitives.

Bennedsen and Caspersen (2006) investigated eight factors that could potentially influence the success of students in a model-driven programming course, only two were found to be significant. These were high school maths result and achievement in the programming coursework. They found no significance in gender, enrolled major or years at university (previous programming experience and team/class were abandoned). These results are different from those found by Ventura (2003) who found that prior programming experience and mathematical ability (defined by high school score) were not predictors of success in his graphical objects-first course.

Allert (2004) has shown that prior experience in computer gaming was a negative factor in predicting success in an introductory CS course, although the reasons are not clear. He suggested that this could have been that the students were not motivated by the course that they had enrolled in or that they spent too much time playing games during their course.

The reasons cited for investigation of factors are interesting, for example Bergin and Reilly (2005) suggest that early diagnosis of likely performance will allow personalised interventions. However, factors known before the semester have generally been shown to be poor predictors. Jones and Burnett's study is an exception in that they demonstrated that spatial ability, as measured in a 3D mental rotation test, was more strongly correlated with results in an initial programming module than with other modules in an Information Technology course (Jones & Burnett, 2007).

A similar visual spatial test was reported to have a small positive correlation with end of semester mark in a multi-national, multi-institutional study that also investigated some other more rarely studied factors. (Simon, Fincher, Robins, Baker, Box, Cutts, Raadt, Haden, Hamer, Hamilton, Lister, Petre, Sutton, Tolhurst, & Tutty, 2006). These were map sketching, ability to articulate a search strategy, and participants approach to study especially with respect to deep learning. All of these factors were shown to have a significant correlation with marks.

Research that was looking for factors that affected programming success are summarised in Table 2-1. In this table an entry indicates that the factor was investigated whereas a blank indicates that it was not. Yes indicates that the

factor was found to have an influence on success, No indicates that the factor was explored but found to have no influence.

Within the innate attributes, gender has most often been found not to contribute to success, whilst learning style has. In relation to previous educational experience, success in mathematics has been found most often to predict success and experience in Computer Gaming is the only factor to have a negative influence on success (Allert, 2004).

The factors investigated seem to be selected by judicious researchers and teachers in an essentially informal manner; although clearly the multi-national, multi-institutional studies require conformity of purpose. The researchers sometimes appear to be largely following intuition that a certain factor, whether it be innate to the student, associated with prior educational experiences, or the particular course, might be an indicator for success in programming.

The nature of the learning style data used in this research is discussed in Chapter 3 but first several different learning styles will be discussed.

Table 2-1. Studies of factors affecting programming success

	Innate attributes		Prior computing and educational experience					Other (mostly course related) attributes
	Gender	**Learning style**	**Maths**	**Science**	**Programm -ing**	**Computer**	**Gaming**	**Other**
Bishop-Clark et al. 1994		Yes						
Chamillard and Karolick 1999		Yes						
Goold and Rimmer 2000		Yes						
Hagan and Markham 2000					Yes			
Byrne and Lyon 2001	No	No	Yes	Yes	Yes			Language score
Thomas et al. 2002		Yes						
Grant 2003	Yes	Yes						
Venturer 2003			No		No(All) Yes (Java)			
Allert 2004		Yes			No	No	Yes Negative	
Ramalingam et al. 2004								Self-efficacy
Rountree et al. 2004								Interacting factors
Bennedsen and Caspersen 2005	No		Yes					Coursework, Enrolled Major Years at university
Bergin and Reilly 2005	No	No	Yes	Yes	No	No		Perception of understanding Comfort level
Pillay and Jugoo 2005	No	No	Yes			No		
Simon et al. 2006								Map Drawing style Spatial Visualisation Articulating search Attitude to study
Jones and Burnett 2007								Spatial ability
Carbone et al 2009								Internal factors / Motivation

2.4 Tools and examining achievement

Since learning to program is difficult for novices, many software tools have arisen from the perceived need to make the process of learning to program more straightforward. A general purpose programming language is an extremely powerful and flexible tool, one that can be used to solve any computable problem. For this reason, many educators have considered that it is not appropriate to introduce novices to such an instrument, but have preferred special purpose languages and/or environments that provide a more structured situation. The difficulties that many students have with the introductory course, also known as CS1, have resulted in some institutions offering a CS0 course. It is worthy of note that such courses do not have the same pressures to use industrial strength languages (Dingle & Zander, 2000). However, this thesis is concerned with learning industrial strength languages and so excludes a myriad of 'experience tools'.

Guzdial (2004, p. 129) claimed that research in novice programming environments was frequently ad hoc, largely because the field lacks a "strong theoretical base". He suggested that we needed to "figure out how to study the ones we have". Gross and Powers (2005) defined three categories that provide a means to classify this research:

1. Anecdotal: may be reports derived from experience of a single study.
2. Analytical: relative to a set of criteria.
3. Empirical: repeatable empirical assessments using observational data.

Whilst the first two have been widely used, the third is considered more desirable.

An important difficulty in researching students during their studies was highlighted by Isohanni and Knobelsdorf (2010) in their study of students' use of a program visualization tool through observation and interview. They wanted to know why some students chose to use the tool and how they used it but discovered that students were reluctant to discuss openly with the lecturer who was responsible for assessing them.

2.4.1 Some early tools

The earliest tools to assist beginning programmers were drawn by hand. Flowcharts and computer programming were for a long time considered inseparable. Goldstein and von Neumann(1947) described processes using a series of interconnected boxes. Flowcharts were considered to be the natural way to begin coding and their use was largely unchallenged until Shneiderman, Mayer, McKay and Heller (1977). They studied the utility of flowcharting as a tool both to write and comprehend programs and discovered no significant differences were found in the achievement of students who used and did not use flowcharts. Although the students who had been using flowcharts did appear to gain some benefit from using them and many students claimed that they found flowcharts useful, the results were not statistically significant.

Pseudo-code

Pseudo-code is an intermediate language between natural language and a programming language. It is has been described as "informal textual representation of a program or algorithm" (Bellamy, 1994) but other definitions are more prescribed. There have been many variations proposed but the main ingredients are that it uses a structured form of natural language using some special words to describe the computational processes. Pseudo-code was adopted by a significant part of the computing community rather than the numerous diagramming techniques which required specialised tools, simply because of the ease of editing pseudo-code with a text editor (Cross & Sheppard, 1988).

Karel the robot

Karel the Robot, was developed by Richard Pattis (Pattis, 1981) and has undergone a number of incarnations which includes Karel++ : C++ and subsequently a Java version (Weber Becker, 2001). Karel inhabits a grid like world and programs direct the movements of the robot, via knowledge retrieved from the robots three cameras. Karel has elementary senses of hearing and touch; he can pick up and put down beepers (which emit a sound). The original versions provided a complete development

environment with a simulator and thus an extremely simple purpose built learning setting.

The benefits of using Karel with novices are that it allows students to concentrate on control structures and their abstractions (Pattis, 1981). Some might argue that this is in fact an undesirable trait because it prohibits learning about variables and data structures. Karel does provide an environment in which confidence can be built. Students can overcome any fears they may have that programming will be too difficult because as they are able to use Karel to solve problems, they will build confidence. Reports on the use of Karel suggest that it provides a motivational setting for students to learn to program, however efforts do need to be made to ensure that students are aware of the possibilities of transferring skills to other programming tasks (Weber Becker, 2001).

Pascal

Although Pascal is a programming language rather than a tool it was certainly designed to make the task of learning to create structured programs easier (Wirth, 1976) Some of the benefits of Pascal for novices are that it has a clear and straight forward syntax, is a strongly typed language which means that errors can be found at compilation rather than run time and uses run time error checking which ensures that array processing is only on valid array elements.

Pascal was used in a variety of programming environments but one of the most widely used was Borland's Turbo Pascal. This IDE included a number of extensions to the language, including a module, similar to that used in Modula-2. This compiler, for its day, was very fast; it reported just the first of any compile errors found and this had the effect that students were able to rely on the compiler for syntax errors rather than careful reading of the code (Jadud, 2005a).

Beyond the features described there was no additional assistance provided to the novice programmer by either the language or the environment. The design of programs was done mainly using pencil and paper and any of pseudo code, N-S diagrams or even flowcharts. In fact at the time it was considered essential to design the program on paper, before approaching the

computer to begin programming – just one step removed from the earlier necessity of entering the program on coding sheets before passing it to a data entry operator to transfer the program to cards.

2.4.2 IDEs for novice programmers

The IDEs used in commercial software development are very complex. They provide many features that are far beyond the requirements of novices and as a result some programming environments have been designed explicitly for beginners. They do not attempt to provide an environment for commercial programming but do provide a simple environment in which students can become familiar with an industrial strength language. The first of the two environments that are considered in this section, Dr Scheme and Thetis, have a slightly different approach to the others in that they also provide a "cut down" version of the programming language. This approach has the benefit that the compiler can provide more direction to the novice programmer than is normal for that language. This process, known as "sub setting", has been evaluated by DePasquale, Lee and Perez-Quinones(2004) and been shown to be effective, in that the novices make equivalent progress when exposed to a full blown environment such as Microsoft Visual Studio, as those that used the professional environment from the start. The remaining two environments are BlueJ and jGrasp, they provide a simplified interface to commercial programming languages; BlueJ to Java, and jGrasp to a variety of languages including C, C++, Ada and Java. They both have evolved from earlier incarnations, Blue and PCGrasp, respectively but in this section only their current versions will be considered.

Dr Scheme

Scheme is a member of the functional programming language family and shares with the more well-known Lisp, prefix syntax. It is often considered difficult for beginners to master because it requires the use of many parentheses. Dr Scheme is a tool that is designed for beginners to learn to program in Scheme (Felleisen et al., 1998) and is probably still the most widely used functional language for beginners. The graphical user interface for editing and executing programs and a "tower of Scheme subsets" are some of the strengths of the environment, along with a static debugger that

checks programs before they are run. The subsets of the programming language align with a common sequence in university courses and selecting the "Beginner" level permits syntax checking to follow particularly rigorous rules, which it may not be possible to identify with the full language features available.

Dr Scheme also provides a Symbolic stepper which shows "every reduction step of a program evaluation", an unusual but powerful feature is the ability to step backwards from run-time errors to find the cause.

Thetis

Stanford University chose to use ANSI C as the vehicle for teaching its introductory programming sequence in 1991. However the change was not without difficulty and "frustration level was often higher as a result" (Freund & Roberts, 1996). Thetis was developed in-house at Stanford largely because of their very large number of students, the Macintosh platform and because they had labour interested in developing the software. The objective was to make learning C easier by a more novice friendly environment. This was achieved by building a C interpreter that provided students with less cryptic error messages and did more error checking. The very short evaluation survey of students suggested that they found Thetis both easy to use and helpful in learning to program.

BlueJ

BlueJ is a purpose built lightweight IDE for teaching Object-oriented programming with Java. It is an environment without many of the tools for commercial programming but does provide some specialised facilities that support the learning of Object-oriented programming (Kölling, Quig, Patterson, & Rosenberg, 2003). BlueJ is a derivative of Blue (Kölling & Rosenberg, 1996a) which was a purpose built object oriented language and development environment for teaching novices object oriented programming.

Unlike Java, Blue was a pure object-oriented language with one of the main differences from Java being that the internal and interface structure of the class were clearly separated in the class definition. Another difference was the syntax of routing declarations which permitted multiple input and output

parameters. Although the language was not adopted, probably because of the requirement from the marketplace to provide for learning industrial languages, the environment of Blue once adapted to BlueJ is now being widely used.

The main input screen provides a simplified Uniform Modelling Language (UML) class diagram that continually emphasises the object-oriented nature of the programs. There are facilities to instantiate objects, execute methods independent of a program main and also to inspect the state of an object. Debugging facilities allow the tracing of programs so that programmers can follow the creation and termination of objects and the ability to switch between the implementation and interface of a class allows the user to concentrate on using class abstractions in order to connect objects.

Kölling and Rosenberg (2001) claim that the facilities of BlueJ mean that a different approach to teaching Java programming should be taken and suggest that this includes:

- Objects First
- Don't start with a blank screen
- Read Code
- Use "large" projects
- Don't start with main
- Don't use "Hello World"
- Show program structure
- Be careful with the user interface

Many of these recommendations relate to the notion of composition and comprehension discussed earlier in that they are advocating that programming should begin by understanding (and maintaining) existing programs rather than creating new ones. In discussion on the final point two suggestions are provided on how to deal with I/O in Java when it is problematic for beginners with the language. One is to use the facilities that BlueJ provides to instantiate objects and run methods and the other is to concentrate on larger program examples where these I/O facilities are provided in the example, the novices work in other parts of the program.

BlueJ provides its users with access to the complete API of Java. When programs are compiled, users are exposed to error messages of the Java

compiler one at a time. Thus programmers are encouraged into a method of working with the compiler that includes fixing one error and recompiling. This can be compared with other compilation processes that provide a list of errors.

BlueJ offers an extension API and describes a means of adding extensions to the core system. Some of the extensions available on the website are a better UML extension and a facility which allows used to annotate programs with the role of variables (Sanjaniemi & Kuittinen, 2005).

jGrasp

jGrasp is a lightweight IDE that provides a number of facilities to assist the novice programmer (Cross & Barowski, 2002). Of these facilities, the provision of control structure diagrams (CSD discussed above) is the most immediately obvious to the user. In jGrasp, CSD is added automatically to source code to assist in the understanding of the structure of the code. The remaining facilities include UML class diagrams for Java, debugging, object viewing via the Workbench and the more recent addition of dynamic data structure viewers (Cross, Hendrix, Jain, & Barowski, 2007). In controlled experiments, it was shown that the dynamic viewer assisted students in coding more accurately and improving their ability to locate "non-syntactical" bugs. Of course the programs selected for these experiments were particularly suited to the use of the data structure viewer.

All the novice programming tools provide a complete programming development environment so that it is not necessary to use the operating system beyond the very basic aspects of accessing the program. In order to use the environment of the programming tools themselves, they all provide a user friendly interface that is relatively easy for novices to use with a minimum of instruction on the environment itself.

2.4.3 Observational studies of novice programming

Jadud's (2006) research concentrated on one type of event, specifically the details surrounding the compilation of programs. Many such studies logged only relatively short and isolated sessions (Judd & Kennedy, 2004; Hundhausen et al., 2006; Jadud, 2006), however Norris et al. (2008) logged behaviour for the duration of programming projects.

Hundhausen (2006) studied the behaviour of novices using an IDE from a HCI perspective, with the aim of building better tools, whilst Jadud (2006) used a grounded theory perspective (Glaser & Strauss, 1967) to explore student activities, when learning to program, with a particular emphasis on the edit-compile sequence.

The edit-compile sequence was also the subject of a single study, in initial (Norris et al., 2008) and extended phases (Fenwick, Norris, Barry, Rountree, Spicer, & Cheek, 2009), in which the stated objective was, to investigate behaviour, especially of students who failed, to encourage behaviours that may be more successful. Murphy Kaiser, Loveland, & Hasan (2009) also recorded and reported on compile errors, but in addition included data on run-time errors and the amount of time spent on programming tasks. The tool used for the data capture, Retina (Murphy et al., 2009), provided recommendations on how to correct the subject's syntax errors. Moreover, this program provided summaries of errors to the teacher.

To investigate how well an environment supported the process of programming by novices, Hundhausen et al. (2006) developed a methodology for researching the temporal aspects of novice programmers' activity. The study had some common elements with this research, in particular the use of a log of student activity. However, it differed significantly in that the log was created using a manual process. The first step taken was to create "model solutions" to the problems that were to be given to novices with particular emphasis on defining the semantic components of the solutions. Then recordings made of the novice programmers at work were scrutinised and the processes used by the novices were coded and finally analysed both quantitatively and qualitatively.

A team of expert programmers created the model solutions. The videos of the students were coded into mutually exclusive categories, which provided the start time of various activities e.g. valid component start, or beginning the editing of a valid component. At this stage, a coded log file with a time stamp was created. In Hundhausen's study supporting statistics were calculated including time-on-task and percentage dead time, the latter being when no events occurred (students may have been thinking).

The processes used by novice programmers were investigated to better understand them and eventually build better environments. The methodology used was both time consuming and subjective and hence, although it provides some interesting comparisons on student behaviour, it requires (as they state) more automation to be effective.

Jadud's (2005b) study was motivated by the common criticism, by educators of novice programmers, that they do not think and expect the compiler to do all the work. In the punched card era of the 1960s and early 1970s, a 24 hour turn around between program compilations was considered reasonable. In the early 1980s, when using batch processing mode on mainframe computers, programmers worked assiduously to remove all compile errors before the next attempt at compile–execute. The fast Pascal compilers introduced in the mid-1980s changed the environment of the novice programmer since they altered the possible modes of work. Compilers that only report the first error, force a difference mode of working.

In Jadud's work, an extension to BlueJ (Kölling et al., 2003) logged the code of students' programs every time a compile was executed. Pairs of compile events were examined so that several attributes could be explored, such as the time between compiles and the amount of change made to the program. Findings included the extremely short time frame between compilations and, on studying the programs closely, how minor were the changes that were made to the novices' programs between compilation attempts. At the extreme were students who would compile, remove a line of code, compile, replace the same line of code and compile again. This "thrashing" behaviour is clearly not conducive to learning, since the programmer rarely sees an executing program and further highlighted a lack of understanding of the syntactic requirements.

A purpose built measure, Error Quotient (EQ) was created from pairs of compile events by scrutinising and classifying the program and any changes made. Some of these features were then used as input to an algorithm devised to produce the EQ. The EQ algorithm penalised students who repeated the same compile error in consecutive compiles, far more than those who made different errors. Importantly it provided differentiation

between students and resulted in a spread of values that was close to being normal. A relatively high EQ was tentatively suggested to relate to poor programming technique whilst low EQ values indicated lowers rates of compile errors and generally more success in programming.

Investigation of the correlation of EQ to exam scores showed a moderate correlation. From this, Jadud hypothesised that if the EQ could be calculated in real-time then it would be useful to the class instructor, who would be made immediately aware of students who were struggling.

ClockIt, was developed to explore the behaviour of students and differentiate between successful and unsuccessful students, especially since there was a mismatch uncovered between the manner in which students and faculty reported software development processes (Norris et al., 2008). The preliminary data were collected using a data logging extension to the IDE during closed laboratories and reported details such as which student wrote the most code or spent the largest amount of time or invoked the compiler the most. Ten types of events were stored in the log file, including project open and close, package open and close, and compilation success, warning and error. The events were time stamped, included project name and, in the case of the package events, the number of files and file size and, in the case of compile errors, the error message and location.

The logged data was analysed via the ClockIT Data Visualizer which provided several views of the event log data including a histogram of the number of events over time and pie charts showing the proportion of successful compilations and program invocations. Norris reported that the system "seems to provide some insight regarding student software development practices", including that the error rates ranged from 66% to 89%. There were no hypotheses either formulated or tested.

A later report on the same project by Fenwick (2009) collected data over a longer period and confirmed Jadud's results (Jadud, 2005b) regarding the most common group of errors made by novice programmers and the common time between compiles. The group provided histograms of some statistics e.g. assignment grade against time spent on assignment. They concluded that "there appears to be a correlation between assignment

success and the amount of time and programming activity" (Fenwick et al., 2009, p. 300); although no such statistics were provided.

In another automated study that investigated novice programmers, Spacco et al. (2005) created Marmoset to collect data from a project submission system. This included detail, such as, the size of the project and the number of unit tests passed, both when submitted by the student and when the program was saved. The resulting analysis was considered complex and the study failed to provide a clear methodology for obtaining meaningful results from this vast data set.

Data, collected over five years, were used to study the behaviour of students (Edwards, Snyder, Perez-Quinones, Allevato, Kim, & Tretola, 2009). This study was motivated by the idea that high failure rates may be due to poor study skills especially in regard to the timing of assignment work by students and hence the scope of the study was limited to the number, size and timing of assignment submission. To create two groups for hypothesis testing, students were partitioned into two groups according to their grades (A/B and C/D/F). The assignment submissions included in the analysis were those belonging to students who did not achieve the same grade in all of their assignments to attempt to uncover the difference in behaviour that caused the difference in the grade. It was shown that the assignments that were awarded higher grades had been started earlier and were submitted fewer times; although there was no significant difference in code size.

Rather than create a log of student activity, an attempt to mine useful data by accessing the Concurrent Versions System (CVS[3]) repositories was undertaken by Mierle, Laven, Roweis, and Wilson (2005). They aimed to locate features of student behaviour and programming that were "predictors of performance". Some 166 potential features were extracted from the available data in three ways. First, temporal aspects of updates and simple counts of the number of revisions were obtained; second, by parsing the code, features such as a count of the control structures were calculated, and finally, a style checker was used to examine the coding approaches used.

[3] a version control system commonly available on Unix systems that records a history of all file versions

Mierle et al. (2005) considered that their study was advantaged by having records of many students undertaking the same tasks. The hypotheses that they were testing, involved studying the effect of work habits and code quality on grades. They used a binary grade variable that identified students in the lower and upper third of exam grades; discarding the middle achievers. Tests were carried out across each feature and the grade. In total they tested 166 features and reported three that were significant, this is consistent with a Type I error rate of 0.05 (described in section 4.8, detailed in Table 4-4. Type I and Type II errors). Two of these involved the lines of code written and the third was the number of times a comma was followed by a space. The conclusion Mierle et al. reached was that students should spend the time needed to complete assignments carefully.

The syntax errors and the time taken by students to correct those errors has surprised researchers (Denny, Luxton-Reilly, & Tempero, 2012). Denny et al. compared fast and slow response groups using hypothesis testing and found no significant difference in the time to respond to syntax errors. Another automated data logging tool, Retina was designed to provide students and teachers with feedback on the types of errors being made and the amount of time being spent collected information on the compile and run time errors made by students (Murphy et al., 2009). The tool produced real-time feedback, as recommendations based on the type of compile errors; this was sent to the student via instant messaging. The tool also reported on these errors as a summary of the class to the teacher. At the end of semester some further data analysis was carried out. This including calculating a number of correlations:

- Time on a given assignment and assignment grade : no correlation
- Overall time for the semester and semester grade : negative correlation
- Number of compilations errors and semester grade : negative correlation

The timing of the compile errors made by students was investigated. This increased after 8pm and was highest between 1am and 4am which resulted in a recommendation to just work during the day and early evening. (Denny et al., 2012)

Table 2-2. Observational studies of novice programmers

Author	Compile / runtime / other	Tool used for data collection	Analysis	Hypothesis formation	Results
Jadud 2005	Compile	Extension to BlueJ logged Code	Created purpose built measure (EQ)	None	Descriptive results of common compile errors and frequencies
Mierle 2005	CVS	Data mining CVS repository. Code parsing and style checking		Each of 166 features vs grade. Using mutual information (a form of correlation)	3 features significant. • Lines of code • Space following comma • And unreported
Spacco 2005	Submission	Marmoset Project snapshot on save and submission	Number of Junit tests passed / failed and static code analysis warnings	None	Indefinite
Hundhausen 2006	Semantic	Manually coded from recording	Manual assessment of semantics matched with expert solution	None	Descriptive
Norris 2008	Compile	ClockIT Event log extension to BlueJ	Visualizer Histograms Pie Charts	None	Successful students have fewer compile errors and write more code
Edwards 2009	No of submissions	Project submissions	Assignments from non-consistent students. Grouped according to grade	Hypothesis that number of submission would be different	Significant difference but no practical difference
Fenwick 2009	Compile	ClockIT Event log extension to BlueJ	Number and frequency of compilation errors and warning	None	Compile error frequency changed over semester. Less “missing semicolon’s” more “unknown methods”
Murphy 2009	Compile	Retina	Correlation between time spent on an assignment and grade. Over entire semester	Descriptive real time feedback of errors. Correlations	Both time and #compiles over entire semester negatively correlated to grades
Denny 2012	Compile / Syntax	CodeWrite	Common errors and time to solve	Hypothesis test compared quartiles speed	Students struggled equally

Table 2-2 summarises the studies that have been discussed in this section. It can be noted that about half of them concentrated on the compilation process and the remainder on various semantic programming issues of programming. The reports of these studies have emphasised novel data collection methods rather than the effect of behaviour on grades. Two studies (Mierle et al., 2005; Murphy et al., 2009) used correlations of various factors against final grade. Murphy considered very few correlations between behaviour and grade but the study concentrated on descriptive feedback in real-time. Mierle investigated many aspects of style in the completed code, rather than the behaviour that was exhibited to create it. The form of analysis done in every case appears to be ad hoc and has often been selected to demonstrate a particular feature of behaviour.

2.5 Learning styles

Since there is a large body of work on the importance of learning styles to academic success, this subject cannot be ignored (Kolb, 1981; Dunn, Beaudry, & Klavas, 1989; Bonham, 1998; Soloman & Felder, 1999; Coffield, Moseley, Hall, & Ecclestone, 2004). Many different learning style models or inventories have been proposed including Kolb's Learning Style Inventory (KLSI) and Dunn and Dunn's model.

Many people consider learning styles to be an important concept in education. There is a view that individuals approach learning with different methods and that they process information in different ways and, that the theory of learning styles can explain these differences. However, there has been some significant criticism as to the validity of learning style theories because of a stated lack of evidence. It was suggested that learners of various stated preferred styles, should be randomly allocated to different teaching methods in order to assess the validity of a particular model (Paschler, McDaniel, Rohrer, & Bjork, 2008). This would be a difficult scenario to adopt and is likely to be considered unethical in most educational settings.

That people are attached to their learning style permanently, is an idea that is considered either completely correct or completely incorrect depending

on the nature of the particular model that is considered. This idea, which at first may appear trivial, is quite pivotal to the manner in which educators should treat learning styles. If learning styles are considered fixed, then it is contingent upon educators to provide learning experiences to suit various styles or possibly, even at an extreme, to counsel individuals that they do not have the learning style necessary to succeed in an area (Coffield et al., 2004). McKeachie (1995, p. 2) warned of a self-fulfilling prophecy arising from the notion of intransigent learning styles that, "some students who have been labelled as having a particular style feel that they can only learn from a certain kind of teaching". He urged that learning styles should be considered simply a preference and that students should be taught strategies to cope with learning situations that they did not find themselves naturally well-suited to. Alternatively, if learning styles are adaptable, both students and educators can work towards a single common approach to learning and teaching or possibly gain benefit from using many different styles to suit the particular situation.

The three learning styles inventories that have been selected for investigation, Dunn and Dunn, Gregorc, and Kolb, are among those more widely applied; all consider learning style to be largely an innate feature of the person.

One way of demonstrating the validity of a learning style inventory is the test/re-test method. If individuals are assessed twice, over an interval of time and their style is identical then this clearly demonstrates that the LSI is more valid than if completely different results were reached (Allert, 2004).

2.5.1 Dunn and Dunn

According to Rita Dunn (1989), "Learning style is a biologically and developmentally imposed set of personal characteristics that make the same teaching method effective for some and ineffective for others." Dunn and Dunn's learning style model has five stimuli or variables that affect how people learn. These are environmental, emotional, sociological, psychological and physiological and each has its own four factors on which students are asked to identify the strength of their preference. For example, the Sociological variable has factors which relate to a student's preference

to work either, alone, in groups and with help or motivation from a teacher. Details of the variables and factors are shown in Table 2-3 (Searson & Dunn, 2001) .

This model is applied by eliciting an individual's preference to the factors. In the Environmental factors for example, some individuals may prefer to work with background music whilst others prefer quiet, some people work better in the morning and others in the afternoon. The idea of the model is to provide a learning environment which matches each student's preferences; although on a purely practical basis it is difficult to see how some these factors can be resolved. Given the number of factors and the potentially large number of combinations of individual learning styles, it is convenient that some preferences have been shown to be linked in certain student groups.

Table 2-3. The Dunn and Dunn learning-styles model (Coffield et al., 2004)

Variable	**Factor**			
Environmental	Sound	Temperature	Light	Design (Seating, layout of room etc)
Emotional	Motivation	Degree of responsibility	Persistence	Need for Structure
Psychological	Global	Analytic	Hemisphericity	Implusive/Reflective
Physiological	Modality	Intake	Time of day	Mobility
Sociological	Learning groups	Help/Support from authority figures	Working alone or with peers	Motivation from parent/teacher

Assertions on the reliability, validity and impact of the Dunn and Dunn model, appear to have largely emanated from those working closely with the model and may lack independence. Searson and Dunn (2001) purport to demonstrate the application of the model. In fact they exposed all students in three groups to different styles of teaching and showed that students learnt more and used higher-cognitive processes with active style, rather than traditional passive teaching methods. Learning style does not appear to have been used in the study although it is discussed in the paper. St John's University, where Rita Dunn is a professor, have many dissertations and

papers which prove the validity of the Dunn and Dunn learning style model. Rita Dunn claims that there are only "three comprehensive models of learning styles", citing her own and two others by Hill and Keefe which are not widely known (Dunn et al., 1989). Coffield et al. (2004) cite several reviews that dispute the validity and reliability of the model. However, despite this, it is used with some authority in schools in the US and many other countries.

2.5.2 Gregorc

Gregorc's Style Delineator (GSD) was first published in 1982. In it he defines four channels or perceptions that he claims define behaviours which indicate how a person learns best. Like the better known KLSI, he defines two dimensions in which learning preferences differ. In GSD these are, Concrete – Abstract and Sequential – Random. Each of these is combined to give four learning preferences which are said to be innate (see Figure 2-1). The GSD was designed to be self-administered. It is a set of ten groups of four words and the individual is asked to rank these as being more or less descriptive of them. Therefore (like Kolb) they are classified as belonging to one of four quadrants defined by the two axes.

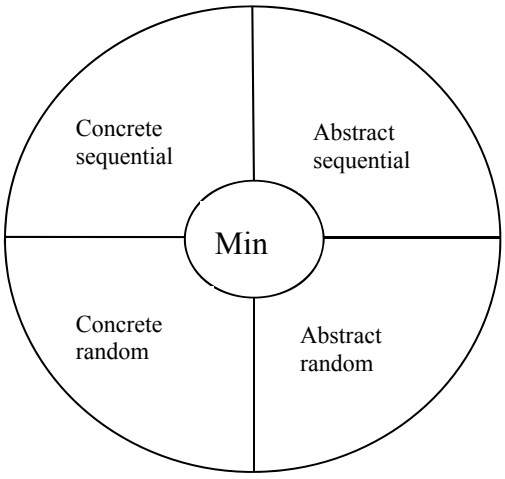


Figure 2-1. Gregorc's four-channel learning-style model

The styles can be summarised as (Gregorc, 1979):

- Concrete sequential: prefer hands-on activities with step by step instructions
- Abstract sequential: have a verbal, logical analytical approach

- Abstract random : like visual methods, group discussion and evaluating personal experiences
- Concrete random: trial and error is a preferred methodology, like to be independent, tend to be impulsive

Despite the use of GSD in a large CS study (Drysdale, Ross, & Schulz, 2001), there are reservations about its reliability and validity. Harasym, Leong, Lucier and Lorscheider (1995) found no relationship between any of the four learning styles and achievement; further they used factor analysis and concluded that the concrete-abstract dimension could not be corroborated.

2.5.3 Kolb

Kolb based his Learning Style Inventory (LSI) on his theory of experiential learning (Kolb, 1984). In his studies of methods of teaching, he noted the preference of some students for certain learning activities and his LSI was the result. He considered the learning process to be the transformation of experience into knowledge through various processes which would ideally be the four stages of his learning cycle (Figure 2-2).

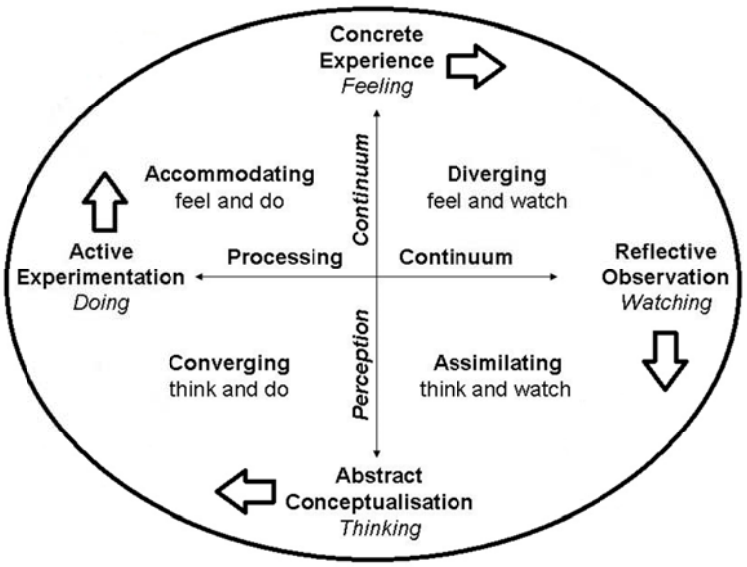


Figure 2-2. Kolb's Learning Style Inventory

Kolb's LSI has been the subject of many research projects. In 2004, these numbered over one thousand and included 104 in Computer Science

(Coffield et al., 2004). Kolb (2000) referred to a learning style as a "differential preference for learning" and there is slightly conflicting evidence on his view on the stability of this style. This has been viewed in the literature as one which may change over time and even in different situations and yet Kolb has claimed that the preferred learning style may be stable over a forty year period (Coffield et al., 2004).

One of the benefits of choosing Kolb's LSI in research is that it has a standardised questionnaire that is available. It is relatively easy to assess an individual's style by answering a short questionnaire which indicates an individual's preferences for learning along two continua that measure perception and processing. The questionnaire requires individuals to rank four potential sentence endings, to partial sentences such as, "I learn best from"; the answers provide a score (positive or negative) on one of the two continua. The Perception Continuum depicts how people think, with Concrete Experience (Feeling) and Abstract Conceptualisation (Thinking) as opposites; whilst the Processing Continuum describes how people do things, with Active Experimentation (Doing) and Reflective Observation (Watching) as opposites. The process concludes by plotting the measures obtained on a graph (Figure 2-2) and an individual's preferred learning style is taken from the quadrant in which they fall. The four resulting styles are (Jonassen & Grabowski, 1993):

- Accommodators: have an accommodating style (feel and do) CE/AC
- Divergers: have a diverging style (feel and watch) CE/RO
- Assimilators: have an assimilating style (think and watch) AC/RO

Convergers: have a converging style (think and do) AC/AE

So using KLSI, an individual's learning style is assessed in two dimensions on the perception and processing continua but is most often reported as a single category.

There is evidence that the learning style as defined by KLSI is not an innate feature of an individual (Geiger & Pinto, 1991) but rather a statement about the currently preferred learning methods.

2.6 Conclusion

The research context has been established in this chapter. Computing Education Research is a relatively new field and one that has not been neatly classified but researchers agree that more empirical research is desirable. Some of the problems with using the scientific method in an educational context were discussed. However, to verify claims it is important to form and test hypotheses.

Observational studies have demonstrated the possibilities of obtaining information from log files and the importance of collecting data of the appropriate granularity. Research has shown that it can be difficult to extract data from students during their studies and there are clearly benefits of automatic logging to obtain accurate data that is collected in an unobtrusive way.

One way of creating new knowledge is through the formation and testing of hypotheses. This is the explorations can be used to explore expansive data and raise questions, those of significance can indicate a direction for further research. Techniques of data mining have revealed methods that indicate that when an attribute is found to be significant, its child/parent attributes may also be significant. Thus, it is possible that interesting relationships lie hidden in the data. The potential problem of combinatorial explosion would need to be dealt with in any investigation that was designed to delve deeply in to the data.

Novices have difficulty learning to program and although the issues have been studied for many years, this is still a problem. Numerous attempts have tried to predict the performance of novice programmers. The factors investigated include those innate to the students, those that relate to prior educational experiences and those that are course related. Previous mathematical success and achievement in coursework have been found to be positive indicators and there is some indication that learning styles may also influence success.

If learning styles are important, then a thorough understanding of the nature of the learning style inventory or model is essential. This will both enable the research process and inform the findings.

The nature of the data used in this research and how it was obtained is described in Chapter 3 and the crux of this thesis, the research model is introduced in Chapter 4.

CHAPTER 3 THE DATA

This chapter describes the data used in this study. The data included the achievement of students from the course of study, their learning style and the observational record of their behaviour as they learned to program.

3.1 Introduction

The data for this research were collected at Murdoch University in Western Australia, in the School of Engineering. Students were enrolled in one of a variety of Engineering specialties offered by the school. All students at Murdoch University also completed a Foundation unit in their first semester of study and this provided the source of the learning style data.

This chapter sets the scene for the research model that is to be introduced in Chapter 4. The three data sets are centred on the student (Figure 3-1); these are:

1. Achievement: the exam score awarded as a result of the students' enrolment in the introductory programming course is the measure of academic achievement.
2. Behaviour: an event log of student behaviour when using P-Coder in the university computer laboratory.
3. Learning style: the students' learning style as assessed using KLSI.

Each of these datasets will be examined to explain the form of the data that they contain. The research model that will comprehensively and systematically explore these datasets is defined in Chapter 4.

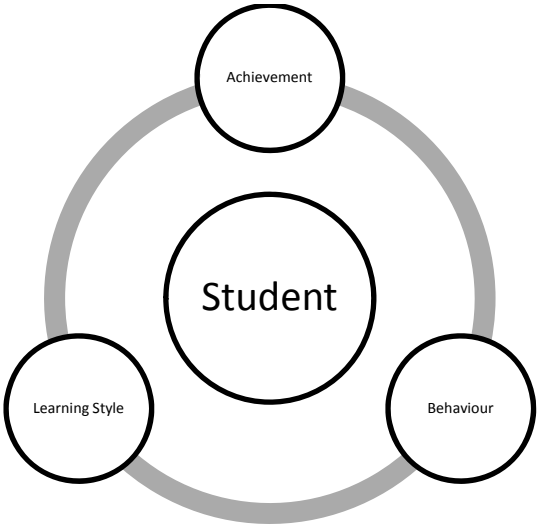


Figure 3-1. The student is the core of the three sets of data

3.2 The students

Murdoch University attracts students with a broad range of academic backgrounds. These include traditional university entrants who completed their school exams the previous year and mature age students (over 20 years old) who are admitted after passing a mature age entry test. Murdoch also provides alternative entry pathways to non-traditional students especially through a summer school program. This program ran as an encouragement to local students to attend their local university and was justified because of the relatively low participation rates in tertiary study. The program ran during the summer and was designed as a bridging course for non-TEE (Tertiary Education Entrance) students. The students had completed either wholly school assessable units or included some VET (Vocation Education and Training) subjects in their high school studies. None of these students had pre-requisites for entry to University let alone into Engineering.

The data were collected over a two year period, during the academic years of 2003 and 2004. All the students were enrolled in the G108 Engineering Computing; an overview of the curriculum is in section 3.3. A total of 108 students were enrolled over the two years; 52 in 2003 and 56 in 2004. Of these, 74 were able to be included in the analysis. The reasons for this are

discussed in section 5.2 when the nature of the data and style of analysis has been described.

There were very few female students enrolled in Engineering and G108 was no exception. There were only four or five female students each year and hence analysis by gender was not possible.

3.3 The course: G108 Engineering Computing

G108 Engineering Computing was a compulsory unit for all Engineering students and completed by most in their first semester of tertiary study. This unit provided an introduction to algorithms and problem solving using the OO paradigm and Java programming language. Rather than use a particular text book, extensive notes and exercises are made available to students in printed form and also online. The single semester (twelve week) course covered considerable ground from the concepts of algorithms and objects, to key principles of languages and grammars and elementary sorting algorithms. The class contact time included two one hour lectures and a three hour computer laboratory session. The tutor was available during the laboratory to mark off completed exercises and provide feedback or assistance as required.

The unit was assessed by the completion of online tests (15%), exercises (15%), a project (20%) and an end of semester exam (50%). The online tests were largely assessing programming knowledge whilst the exercises combined that knowledge with process. The project provided the students an opportunity to hone their problem solving skills, whilst applying the acquired process knowledge. Finally, the exam had several sections that covered a range of domain skills and knowledge.

The average score on the various types of assessments varied considerably. Students achieved generally higher marks in the projects than the tests and both of these were considerably higher than exercises and exams. This pattern was very similar in the two years that data was collected although all scores were slightly lower in 2004 than 2003 (Figure 3-2). It is not clear whether the variation in scores was because of more difficult assessments,

lower achieving students or stricter marking but in any case the differences were small and certainly not statistically significant.

The nature of each of the assessments was quite different; the tests were conducted online and consisted of a series of multiple choice questions. The exercises were small programming problems and each required that a complete java solution be created from a brief description; students were expected to complete at least two or three each week. Students were presented with the project in the seventh week of semester and it was anticipated to require considerable effort over several weeks of the semester. The project was presumed to require that students spend some time to understand the problem before beginning the design and programming. The end of semester exam had four sections. These were finding and correcting errors in code snippets, interpreting algorithms, writing program segments and finally a range of questions testing OO knowledge and some general programming theory. The section requiring students to write program segments resulted in an average lower mark than the other three sections that were approximately equivalent. This result resonates with previous research that indicated students find writing code a difficult task to master (McCracken et al., 2001).

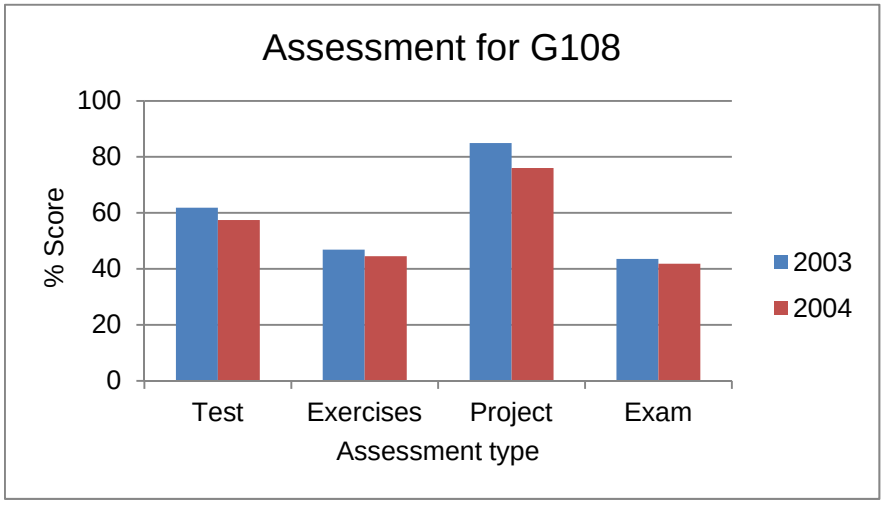


Figure 3-2. Mean scores for the assessments

There are several measures of achievement that are available, including tests, assignment and project marks. However, assignments and projects are not completed under controlled conditions and consequently there is a possibility of collaboration amongst students. Thus, there is a risk that these

marks do not measure a student's own achievement. Since the tests were held during the semester, they did not measure learning outcomes at the end of the semester.

The exam was conducted at the end of semester and can therefore be considered a summation of the semester's work, although there is some concern that students who suffer from exam anxiety would not perform at their optimum level. However, this remained the best available measure of achievement and was therefore selected.

3.4 Students' learning styles

The selection of Kolb's LSI in this research was largely pragmatic (but still broadly justified in Chapter 2). It was selected because at the university used in the research, the concept of learning styles was introduced to students in a foundation unit and the university had arranged to use the Kolb LSI as a means to assist students to understand styles of learning. KLSI was administered by either an online or paper questionnaire.

Section 2.5 has shown that Kolb's LSI was indeed a reasonable choice; at least because this instrument has also been used in several other studies and so provides for useful comparison. Loo (1999) suggested that this instrument remained effective, despite some problems.

Perhaps, not surprisingly, Mainemelis, Boyatzis and Kolb (2002) showed that the majority of studies supported the use of KLSI. However, its use as a guide, to allow matching of the most effective teaching methods with certain students, has been shown to be very limited (Smith, Sekar, & Townsend, 2002). Kolb himself suggests that students should be exposed to a variety of learning modes and not simply those that they might prefer.

The KLSI of several groups of students and staff were reported by Fowler et al. ({Fowler, 2003 #397}). Most of the G108 1st years were also the subjects of this research. The learning styles of the engineering students were diverse, and covered all categories, (Figure 3-3 and Table 3-1). The staff were *assimilators* and *converger*s in a greater proportion; Kolb (1984) suggested that engineering is a good career area for *convergers* and that teaching appropriate for *assimilators.*

Table 3-1. KLSI cumulative results (Fowler, Campbell, McGill, & Roy, 2003)

Clients	No. of Clients	Accommodator	Diverger	Assimilator	Converger
Engineering 1st year Students	126	8%	18%	33%	41%
Engineering Staff	12	0%	17%	41.5%	41.5%
General Arts & Commerce 1st year Students	198	13%	13%	47%	27%
Year 12 all students	112	26%	10%	44%	20%
Computer Science, IT 1st year Students	66	5%	12%	56%	27%
G108 1st years 2003 only	48	4%	8%	42%	46%
4th year Engineering students	29	3%	7%	40%	52%

This study did not collect information on teaching style, but if teachers base their teaching on their own preferred learning style then the potential mismatch between the preferred learning styles of the staff and the students may be important (Table 3-1). Felder (1993) argued that students whose learning styles are compatible with the teaching style adopted within a course tend to retain information more effectively, obtain better grades and maintain a greater interest in the course. If so, then the diversity of learning styles from the students suggests that flexibility in teaching style is of considerable importance.

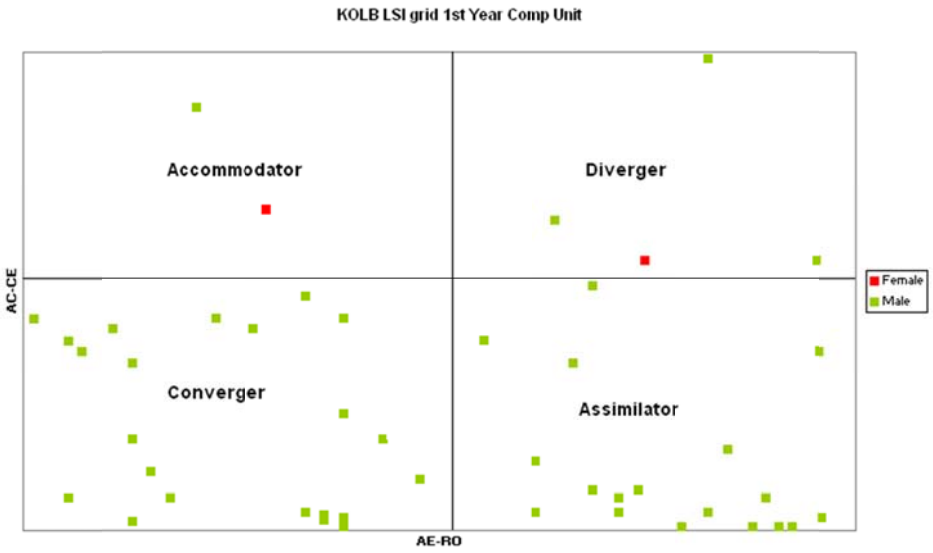


Figure 3-3. KLSI distribution for programming students (Fowler et al., 2003)

3.5 The software design tool: P-Coder

P-Coder is an IDE specifically for novice programmers (Roy, 2006). It provides facilities for the design, development and testing of programs but expressly does not provide all the facilities of commercial environments (e.g. Eclipse and Netbeans) to maintain a simple interface. The raison-d'être for P-Coder is to provide additional assistance to novice programmers in the early stages of learning to program. P-Coder aims to give scaffolded support as beginners stumble through their early attempts at problem solving but it is also flexible enough to remain a useful tool as each individual becomes more competent.

As discussed in Chapter 2, many novice programmers struggle to come to grips with the syntax of a programming language, whilst also finding it difficult to piece together the computational elements required to construct a working program. P-Coder has features that assist in both these areas and others, as will be explained. Since in the programming process, there are different facets of the program that are important at distinct stages of development, P-Coder views provide alternate modes of working and facilities with which to create and explore the program. The views are Designer, Code, Class, Object and Module views.

3.5.1 The Designer and Code views

In Designer view, P-Coder provides the facilities to construct a graphical (tree) representation of a program from a series of components; these components are illustrated in Figure 3-4 and Figure 3-5 and can also be viewed at the left hand side of the screen image in Figure 3-6. Icons can only be placed in valid positions in the work area which provides some guidance to the user on how to create a valid program. The main option, when adding an item, is to either extend the tree or include a sub-tree. The result is a tree structure representation of a program, where each node is a structural, data or computational program element.

Structural elements include packages, classes, data and methods; whilst computational elements include those of sequence, selection, iteration and recursion. Class data behaves slightly differently than the others, since it is added automatically along with Class node, since P-Coder assumes that all classes will have some data. The local data for methods, must appear at the start of a method but is added, if required, to the method node.

Icon	Description
	Package
	Class
	Class data
	Method
	Local data

Figure 3-4. The P-Coder icons for structural and data elements

Icon	Description
	Sequence
	Iteration
	Selection
	Else

	Switch
	Case
	Recursion

Figure 3-5. P-Coder icons for computational elements

When the icons are first added to the tree they are in skeleton form (Figure 3-6) but even at this preliminary stage the equivalent code view of the program can be inspected (Figure 3-7). The correspondence between the Designer view and the Code view is clear.

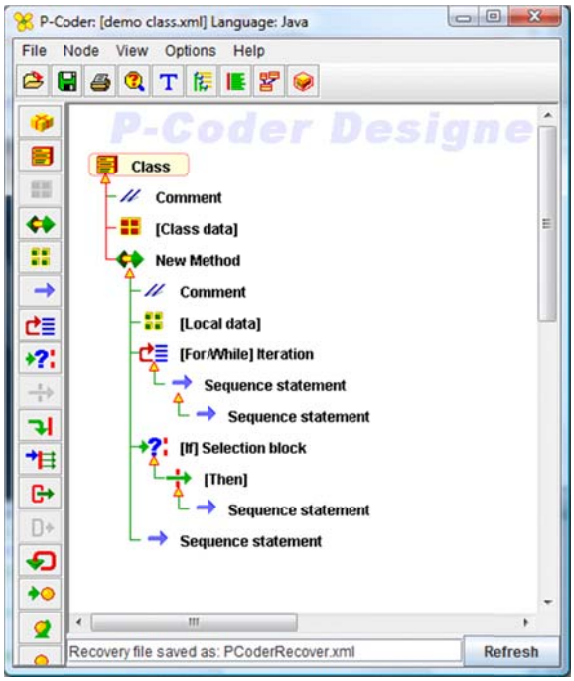


Figure 3-6. A P-Coder skeleton program tree

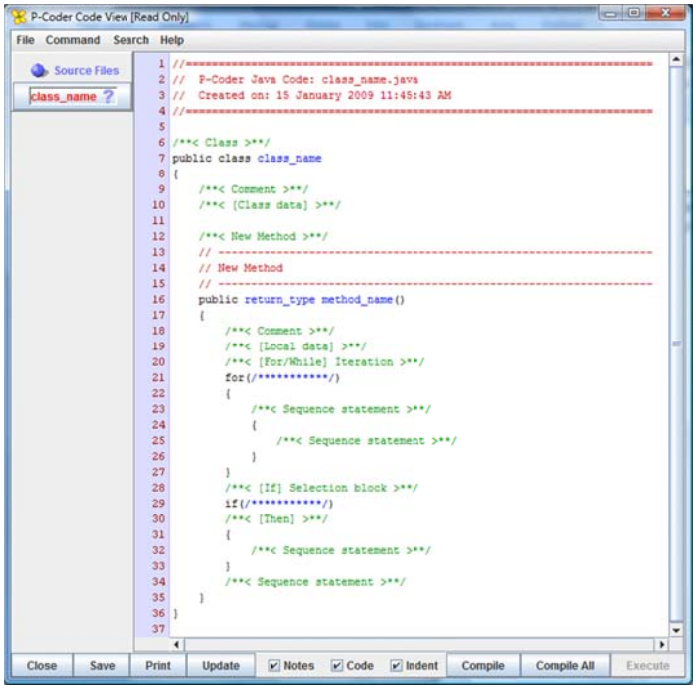


Figure 3-7. The P-Coder skeleton program from tree in Figure 3-6

Figure 3-7 reveals the code that is automatically created from simply selecting icons and placing them in a program tree; although it is rather unlikely that any programmer would spend time looking at the code at this stage in the design. The normal process would be to add some detail to the design before generating the code; in other words writing the pseudo code before attempting to code the program. This detail is written into the skeleton program tree by right-clicking on an icon description.

Once an attempt has been made to complete the pseudocode, many algorithms are clearly identifiable; this is demonstrated in Figure 3-8 which presents a question from an exam paper.

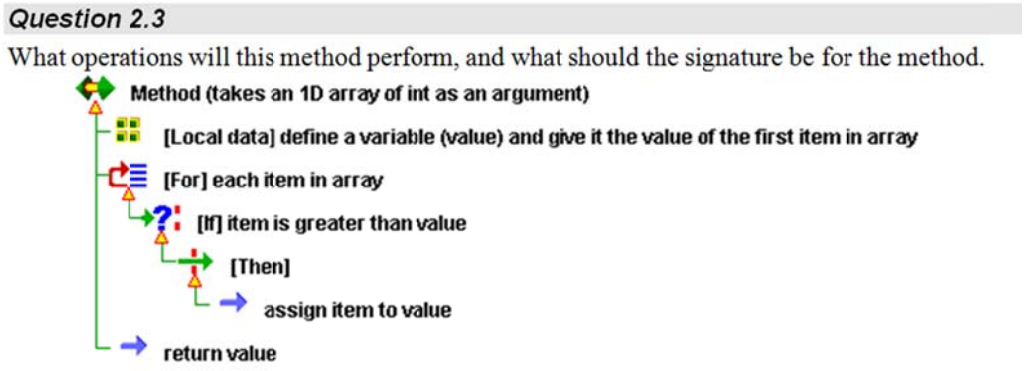


Figure 3-8. An exam question that demonstrates an algorithm in pseudocode form

The possibility to discuss algorithms and other structural aspects of the design of programs at this early stage is one of the benefits of P-Coder.

Novice programmers can learn about the importance of the exact placement of code with the assistance and clarity of the visual cues in the tree structure, without the complication of having to consider the syntax of a programming language. This problem is still to be faced but help is provided in the form of dialogue boxes that contain prompts and text area such as that in Figure 3-9.

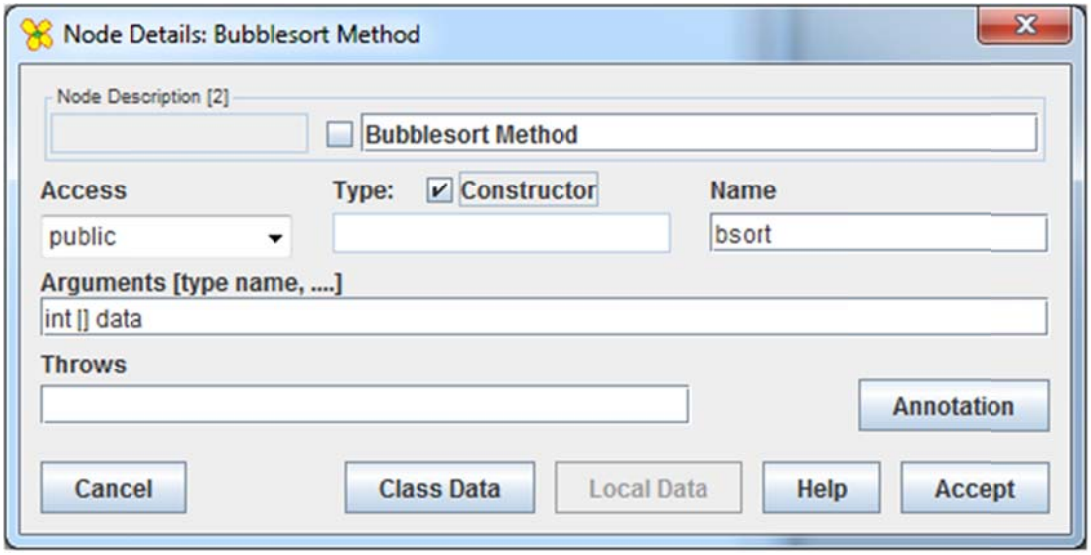


Figure 3-9. Node details for a method header

A user progressively writes a program by completing a series of dialogues that prompt for the required program components. This process provides more direction to the user when compared to entering code into a blank window. The prompts and options available in the dialogs are designed to guide the novice programmer around some of the common pitfalls of novice programmers. One example is that by ticking a box (Figure 3-9), in the node details dialog which identifies the method as a constructor, the user is precluded from the option of entering a return type.

3.5.2 The Class view

The class structure of an OO program is very important. OO programs should be seen as a collection of cooperating objects and so it is very helpful to envision the relationships between the classes from which objects will be instantiated. P-Coder provides a UML styled class diagram in the Class view for this purpose. Classes can be created, viewed and altered from within the Class view.

Three versions of the Class diagram are available; each shows a different amount of the details of the class. The most limited version is very similar to that provided by BlueJ (Kölling et al., 2003), whilst the medium view (Figure 3-10) also exposes the fields and methods of the classes including visibility and a complete method prototype.

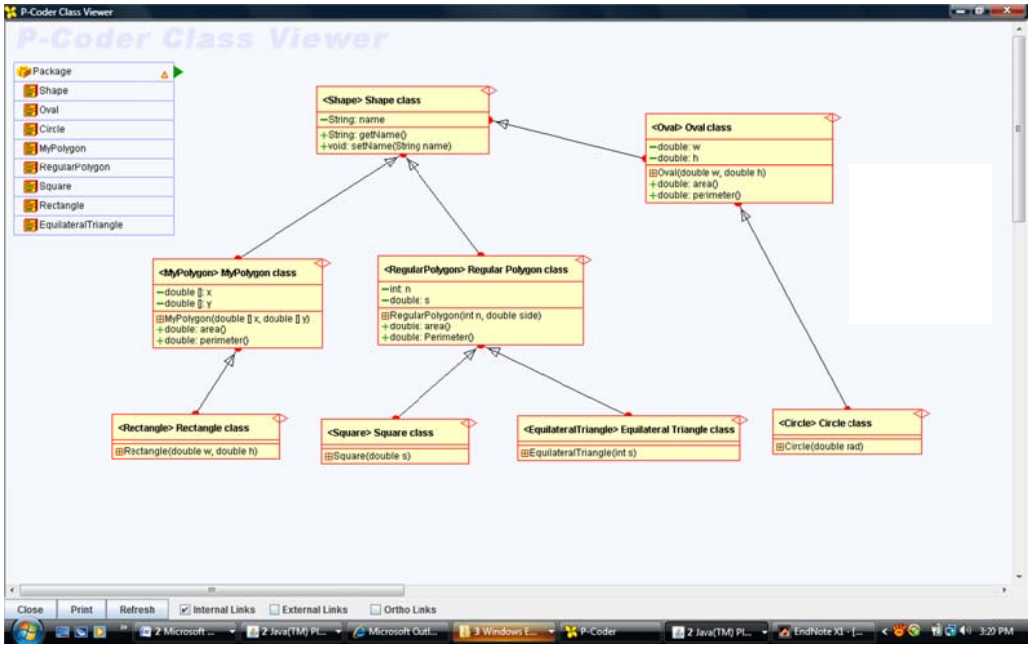


Figure 3-10. P-Coder's Class view

It is also possible to begin the design of a program in the Class view. It could be considered the natural place to start, since this view permits the establishment of classes, associations, data and methods which are normally the first elements to be defined. There are commonly three types of association between classes and each of these can be distinguished in the Class view. The relationships are:

- Extends: to implement a hierarchy or sub and super classes,
- Implements: used when a class implements an interface,
- Uses: when a class uses the facilities provided by another.

These associations are automatically added to the view and this maintains equivalence between the code and the class view.

3.5.3 The Object view

Novice students often find the concept of an object, and how it relates to a class, difficult to grasp. It has been suggested that an environment that

allows students to create, interrogate and test objects, will assist in the assimilation of these concepts (Kölling & Rosenberg, 2002). It is necessary first to create objects, as instances of a class; this then allows the internal state of objects to be inspected and also the behaviour of the objects to be exercised through method evaluation (Figure 3-11).

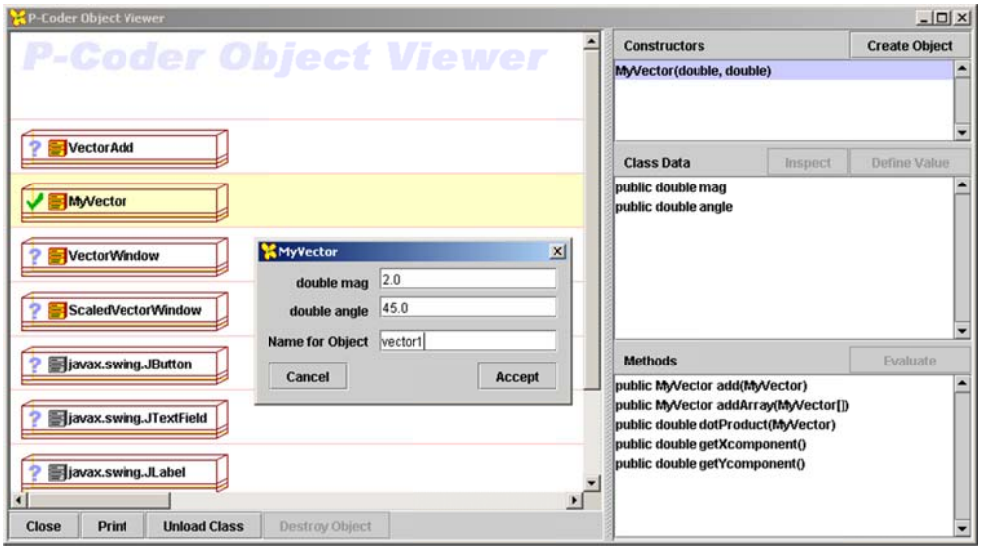


Figure 3-11. The Object viewer: instantiating an object

P-Coder was used at Murdoch University between the 2003 and 2005 in two one-semester courses in Engineering Computing. Anecdotal evidence from students indicated that they appreciated the scaffolded help that P-Coder provided.

3.6 Recording student behaviour

Fisher and Sanderson (1996) considered the requirements for recording from several perspectives with potential relevance to Human-Computer-Interaction (HCI) research. They suggest that:

- the behavioural tradition is served by objective questions that can be answered using the scientific method,
- cognitive researchers tend to favour verbalisations that can be triangulated with automated logging, this allows the study of the intent of the user,
- social researchers tend to prefer qualitative data and use a wider variety of input possibly including verbal and physical gestures.

Automated logging provides a precise record of events and is an efficient means by which to collect data, although it is unlikely to provide as rich evidence of events as the other methods which require human interpretation (Renaud & Gray, 2004). These are discussed in more detail later in section 3.7.

This research seeks knowledge of the behaviour of novice programmers as they learn to program in their first semester of tertiary studies. The data required was a comprehensive collation of behaviour over the duration of the entire semester, rather than a short snapshot of a single session. Thus the data required had to be collected in a live environment that did not require complex special purpose equipment. Automated logging of event data was appropriate to collect this quantity of data in a manner that would not impact on student behaviour. An event log provides a specific type of data, an event "has its own time and location and occurs once" (Kaneiwa & Tojo, 2005). The event should be identified by the user and time to be uniquely identifiable.

The measure of behaviour requires that the precise time of each student activity is known. This will allow the timing of events and hence the duration of each event to be identified. The only sensible model to do this in a timely and cost effective manner is to record events using computer software and store the data in a log file. There are limitations to the temporal data because there are many student behaviours that are not recorded and student actions between events cannot be known; for example, if an event appears to take a long time this could be for several reasons including:

- Thinking time, possibly reading notes or texts
- Switching to other software either related to the problem or not
- Chatting, day dreaming or using mobile phone

3.7 Collecting the data

P-Coder provides for automated event logging of user activity and provided a practical approach to recording the behaviour of novice students in the university computer laboratory. This method allowed student behaviour to

be recorded for the entire semester in a manner similar to that used by Jadud (2006).

If P-Coder's activity log is activated then a user specific identifier (ID) is required to be entered at start-up. The ID is obtained by running a separate utility that creates IDs either singly or from a data file of user names. Each logged activity record will then be readily identifiable and this will permit data to be collected in a single file.

The common data or log file could be stored on a computer network server but this would require users to have write access to a drive. A more easily maintained system, if a web server is available in a university laboratory, is the use of a CGI script on the web server. A message from the user's P-Coder application is sent to the web server and each message is added to the log file. The data for this study was collected in this manner. The data collected required processing before it was ready for analysis, including differentiating users, sorting and filtering. This was done by creating a series of data files, using a purpose built Java program. The statistical analysis on these files used the statistical tool,Analyse-It, an extension to Microsoft Excel.

3.8 The data structure

The P-Coder log files are text files and contain records that are created as the student interacts with the software; the records are at a relatively high level of granularity. Jadud (2006) collected data specifically at the time of compilation of programs by students, this included snapshots of the programs and permitted extremely close investigation of the compilation and included the syntax of the code at this time. The P-Coder log file does not store this level of detail; however, it does record information regarding all events that are carried out by the student as they use P-Coder.

The four basic types of events are:

- Model operations such as loading and saving
- Design operations such as adding, deleting nodes, compiling and executing

- Viewing operations such as opening and closing and manipulating diagrams
- Miscellaneous operations like printing

In addition to the type of event, the records are also coded to allow differentiation of the P-Coder view in use at the time. A complete set of views, the associated event codes and some examples of the events is listed in Figure 3-12.

P-Coder view	**Event code categories**	**Examples**
Start / quit	0 – 999	Start and quit
Designer	1000 – 1999	Events created include structural editing, such as adding a method to the program or sequence node to the algorithm
Module	2000 – 2999	Permits other modules (programs/methods etc) to be opened and viewed or copied into the current model.
Class	3000 – 3999	Add and delete classes as well as resize class to improve the format of the UML class diagram
Code	4000 – 4999	Code view events include updating code, compiling and executing the program
Object	5000 – 5999	Object view methods include loading class files, creating objects and executing methods

Figure 3-12. P-Coder views and event code categories

The event records also contain other information including the project that is current, a description of the event and in many cases some further information that provides greater insight into the programming activity that is being undertaken. The additional information will be explained in a brief outline of the events in each view.

When the student is using Designer view, they are working on a model of the program that is tree structured (Figure 3-6) and events include structural editing of the tree, such as adding a method node to the program or sequence node to the method. The additional information in most of these operations is the node type which indicates whether the node in question is a

sequence, selection, or iteration node for example. The full range of node types is listed in Table 3-2; the coding of node types is also used in the Class view.

In Module and Code views, the additional information is the name of the module or file. In the Object view, the class name is the additional information, when it is used. This is pertinent since this view permits students to create and inspect objects and also execute methods.

Table 3-2. Node Type assignments

Node Type Number	Node Type Description	Node Type Number	Node Type Description
0	Program	10	Switch
1	Package	11	Case
2	Class	12	Default
3	Method	13	Class Data
4	Sequence	14	Local Data
5	Iterator	15	Comment
6	Selector	16	Try
7	Then	17	Catch
8	Else	18	Finally
9	Recursion		

Figure 3-13 shows a small segment of a log with records from five students collected over a few seconds during a computer laboratory session. The student IDs have been partially obscured (by Xs) but it is clear from the first two records that the students were both working on exercise 4. It can be noted that the time stamps of the records are not completely in order, so an important operation before attempting to look for particular sequences of events is to sort it. One of the records in the 4000 or Code category is a program compile but the remaining four students are working in Designer mode and creating 1000 category events.

Looking yet more closely at the data and following the activities of a particular student (HXXXB1386):

1. The first record is performed at 53 seconds past 3:15 pm in the afternoon of 5th March. The project is from section 7, exercise 4 and

the student whilst working in Designer view has just updated some details in a dialog box.

2. One second later the student adds a note (comment) node to a selector (if statement).
3. Five seconds after this the student cuts a sequence tree node.
4. Two seconds later a details dialog of a then node is opened.

HXXXB1386	2003.05.05.15.13.53	2	section7exercise4.xml	1069:Designer	Details updated
BXXXXK2688	2003.05.05.15.13.56	2	Exercise4.xml	1069:Designer	Details updated
YXXXXXS3634	2003.05.05.15.14.45	1	emtehan.xml	1009:Designer	New:6
CXXJ7978	2003.05.05.15.14.21	1	assignment1.xml	1071:Designer	Note:2
SXXXXR3721	2003.05.05.15.14.04	1	ex91.xml	4031:Code	Compile:draw.java
HXXXB1386	2003.05.05.15.14.04	1	section7exercise4.xml	1039:Designer	Cut:4
YXXXXXS3634	2003.05.05.15.14.52	1	emtehan.xml	1009:Designer	New:6
HXXXB1386	2003.05.05.15.14.06	2	section7exercise4.xml	1072:Designer	Opening:7

Figure 3-13. An example of records from the P-Coder log

3.9 Examples of analysis from the log

The temporal aspect of the log reveals the behaviour of a student as he/she is using P-Coder; for almost an hour in Figure 3-14. Since the P-Coder log uses numeric values to identify the events; this facilitates graphing such as that shown. The P-Coder view and specific event can be identified given knowledge of the numeric codes used (Figure 3-12).

For almost half of this time, the student is oscillating between designer (1000 category) and code (4000 category) views. The student is in fact mainly editing nodes (to alter the detail of control structures etc.) and compiling his/her program, he/she does not appear to obtain a clean compile because the program is not executed. There are a series of events in the class

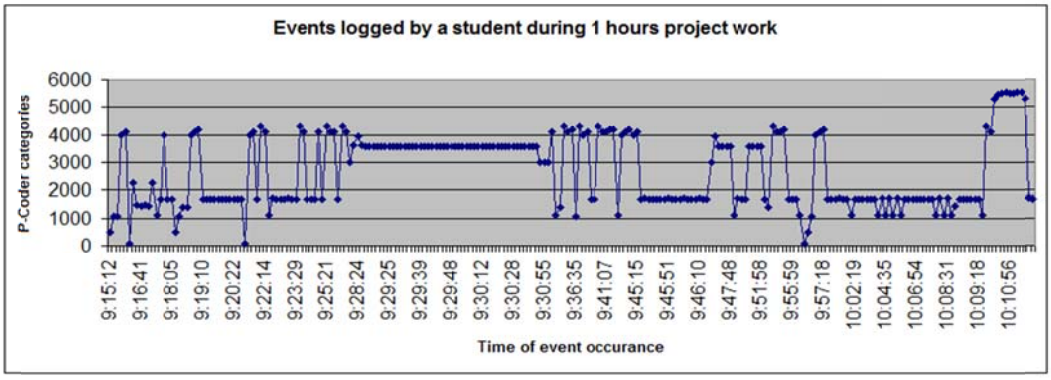


Figure 3-14. Events logged during a one hour work session

view (3000 category) where the student is exploring the relationship between classes and, after a more sustained period of design events nearing the end of this session, the student moves to the object view, where objects are created and methods executed. This figure provides a very functional insight into the student's activity, but it does not reveal whether this is the student's normal mode of working or whether this pattern of events is helpful in aiding their learning.

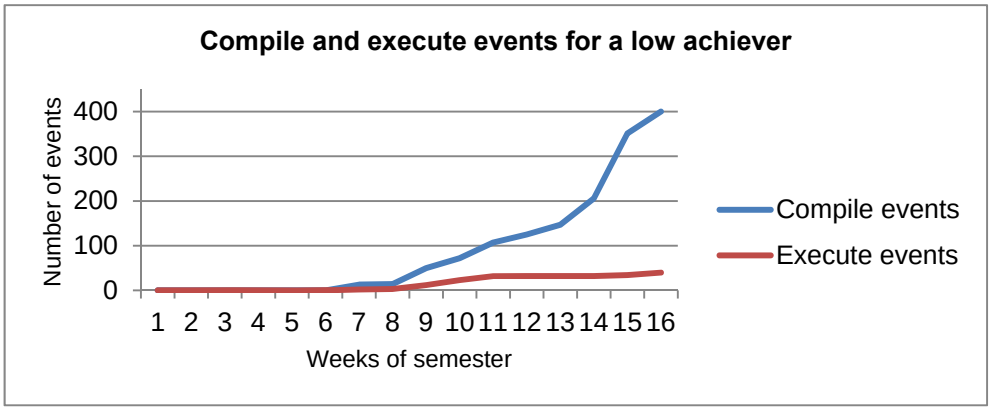


Figure 3-15. Cumulative events from a low achieving student

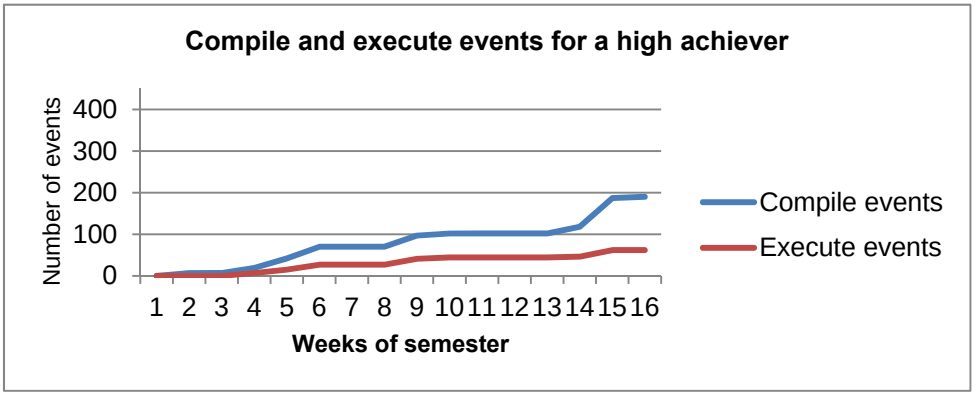


Figure 3-16. Cumulative events from a high achieving student

Another example of the information that can be extracted from the log is shown in Figure 3-15 and Figure 3-16. The graphs provide a taste of a longer term temporal view of student behaviour. Specifically, they show the cumulative number of compile and execute events during the weeks of the semester. It is noticeable that both students do very little in this regard for the first few weeks. This is because at this time the course required them to develop algorithms and they had not yet learnt about the specifics of coding, compiling and executing programs. For both students, the number of compile events is considerably larger than the number of executes, presumably because several attempts at compiling are made before an

executable program is achieved. The difference is much greater for the low achieving student. After week 11 of the semester they barely execute a program (it is possible that they are using the object view instead and executing methods independently of the entire program) and the high frequency of compiles in the last weeks seems to represent a level of desperation in attempting to obtain a clean compile. This can be contrasted with the high achiever who appears to have completed the requirements of the practical work for the semester and can be assumed to be now studying for the exam.

One more view of the data for the high achieving student is shown in Figure 3-17. The time in minutes that the student spent using P-Coder in the laboratory for each week of the semester is shown. Also revealed is the period, in blue, when the student was clearly working on their project, while in the time shown in red, the student was working on a variety of other tasks. The project for the semester was developing a Snakes and Ladders game, for this student this project is clearly visible in the program names used. This student had six versions of the program. It is possible that this is a slight underestimate of the actual time spend on this project because it may be that some of the red area on the graph should be blue because of the possibility of not naming a project immediately. It is clear that despite the fact that the project was made available to the students in approximately week 9 of the semester, the student was doing more work on it in the labs during weeks 13 and 14.

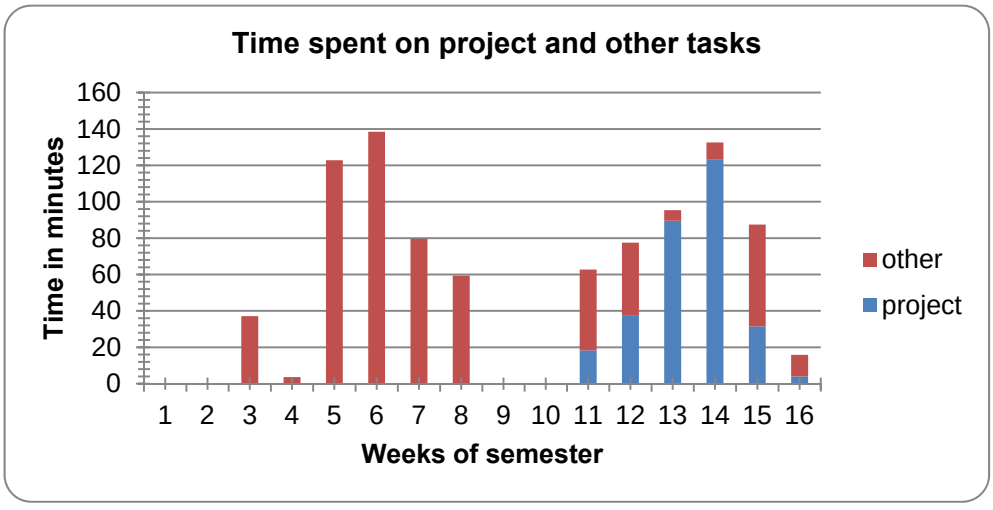


Figure 3-17. Time spent on project and other tasks

3.10 Learning style and P-Coder use

This section provides an alternative view of some of the data to which the research model is later applied. This was collected from the activities of G108 students from 2003 and 2004.

The proportion of events that the average student of each learning style executed over the course of the semester is shown in Figure 3-18. The largest proportion of events is in Designer view for all learning styles, followed by Code then Object view; there is a small use of Class view with no visible use of Module view. The pie charts are arranged to match the four quadrants of Kolb's LSI; this has the vertical Abstract/Concrete axis and the horizontal Active/Reflective axis. Although the differences are small, it should be remembered that these are averages over the students, so some observations can be made of the diagrams in relation to these axis and KLSI.

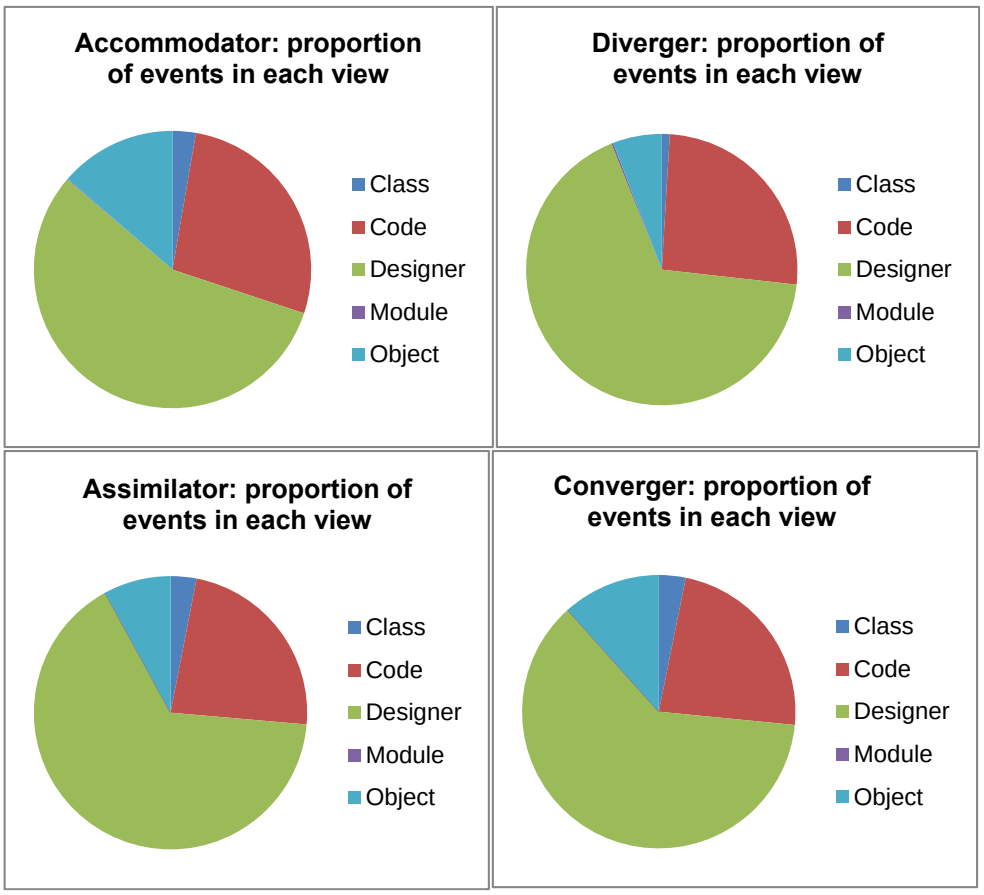


Figure 3-18. Learning style and use of P-Coder views for G108 students in 2003 and 2004.

Accommodators, and to a slightly less extent Convergers, use the Object view in a greater proportion than both Divergers and Assimilators. The Object view is used to instantiate objects, investigate the contents of objects and execute class methods. These operations can all be seen as a form of active experimentation, in the very passive world of the (especially unfinished) program and this is precisely what Kolb's LSI would predict.

Accommodators and Divergers can both be seen to use the Code view for a greater proportion of events than Convergers and Assimilators, this provides access to the code of the program, to observe, compile and execute it. Programmers who preferred concrete experiences rather than abstract concepts might be expected to have a tendency to favour this view rather than the abstraction of the program in the P-Coder tree.

3.11 Summary

This chapter has described the data to be used in this research. The exam score has been selected as the best available measure of achievement, as it is obtained at the end of the period of study and under controlled conditions. The learning style of the students is classified using the KLSI. This is a two dimensional measure along the two continua, perception and processing.

The final data set is a log of events automatically recorded as students programmed in the university laboratories throughout the semester of study. This is a comprehensive and rich data set that can be filtered, analysed and summarised to provide many different types of information; a few of these have already been illustrated. The breadth and depth of this data will be revealed in the application of the research model in Chapter 5, but before that Chapter 4 will introduce the research model that is designed to explore the relationships in this data.

CHAPTER 4 RESEARCH METHODOLOGY AND DESIGN

The main contribution of this thesis is the development of a research model that provides for a systematic empirical investigation into data, such as that discussed in Chapter 3. This chapter presents that model. Following a justification of the selection of the research approach and an introduction of the model, important terms are defined. An explanation of the structure of the research model then precedes a definition of the algorithm that drives the research and the manner of presenting the results. Next is a description of an experimental application of the model in the domain that was introduced in Chapter 3; including transforming the research questions into an operational method and describing the statistical tests that are used. The chapter concludes by considering the limitations of the research model.

The research context was established in Chapter 2. This provided both the foundation and perspective for this research model. It was shown that novices find learning to program difficult and there continues to be a lack of empirical research in CSE. It was proposed that automated logging is an effective method of recording behaviour without disruption to the subjects. Processes that can be used to form new knowledge through hypothesis testing were introduced, along with a suggested means of locating potential new relationships in the data hierarchy. The largely ad hoc nature of the selection of factors, to test whether they influenced success in programming, was highlighted and finally it was noted that learning style may be an issue in affecting student achievement.

4.1 Selection of the research approach

A suitable research approach must be found to answer the research questions. Galliers' (1990) taxonomy identifies various approaches with their appropriate application areas and is reproduced in Table 4-1.

The main research question of this thesis (stated in section 1.3) is "Can empirical records of student behaviour contribute to an understanding of how students can achieve better learning outcomes?" In its initial form, clearly an observational rather than interpretational approach was required and the object of enquiry (Table 4-1, highlights in orange) was the individual (student). The question did not require theorems to be solved and, although a survey would provide empirical records, it was shown in Chapter 2 that automated records are more likely to provide a precise record of behaviour. Forecasting, simulation and role playing are indicated as appropriate but these are at the interpretation end of Galliers' research mode spectrum and Chapter 2 also showed that more empirical research was needed and hence was sought. The remaining methods in this part of the taxonomy are laboratory experiment, field experiment and case study; with case study identified as least appropriate when the object of study is the individual.

A major weakness of laboratory experiments is the "limited extent to which identified relationships exist in the real world" (Galliers, 1990, p. 161). In order to contribute to an understanding of student behaviour, a field experiment is more appropriate and further a dataset was available that was collected using a process of automated logging (section 3.6).

Table 4-1. A taxonomy of IS research approaches (Galliers, 1990, p. 168)

	Modes for traditional empirical approaches (observations)							**Modes for newer approaches (interpretations)**			
Object	**Theorem proof**	**Laboratory experiment**	**Field experi-ment**	**Case study**	**Survey**	**Fore-casting**	**Simu-lation**	**Game /role playing**	**Subjective/ argument-ative**	**Descrip-tion/ interpret-tive**	**Action research**
Society	No	No	Possibly	Possibly	Yes	Yes	Possibly		Yes	Yes	Possibly
Organization group	No	Possibly (small groups)	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes
Individual	No	Yes	Yes	Possibly	Possibly	Possibly	Yes		Yes	Yes	Possibly
Technology	Yes	Yes	Yes	No	Possibly	Yes	Yes		Yes	Possibly	No
Methodology	No	No	Yes	Yes	Yes	No	Yes		Yes	Yes	Yes
Theory Building	No	No	No	Yes	Yes	Yes	Yes		Yes	Yes	Yes
Theory Testing	Yes	Yes	Yes	Possibly	Possibly	No	Possibly		No	Possibly	Possibly
Theory Extension	Possibly	Possibly	Possibly	Possibly	Possibly	No	No		No	Possibly	Possibly

As this research has progressed, the ad hoc nature of some previous research became apparent (see section 2.3), thus guiding the focus of the research to an emphasis on the development of a model. This also changed the emphasis from the individual to Galliers' theory building (Table 4-1, highlights in green). The requirements for a model include that it will provide the framework for analysis in a repeatable process. Galliers identifies that a case study approach is entirely appropriate for theory building; so the domain of novice programmers remained suitable although the research mode was altered.

The next stage in the research process was naturally, testing the model (theory testing) and Galliers indicated that three modes theorem proof, laboratory and field experiments are appropriate approaches to take (Table 4-1 highlighted in purple). Since the main research question was seeking to use empirical records of behaviour to form an understanding of how to improve learning outcomes, it was fitting to select a field experiment in the original domain to test the model.

4.2 Introduction of the model

The research model will provide for thorough quantitative investigations into the relationships between pairwise datasets, where the datasets each contain measures of an attribute of the same object (student). It is suitable for research that seeks to find interesting[4] relationships in the data.

The model specifies a control process that guides the research to systematically investigate relationships in the data using hypothesis testing. The model is designed to use hypotheses that require pairs of attributes and the control algorithm maintains this requirement. The hypotheses are selected so that most potentially interesting relationships in the data are presented for testing. The research process is directed to continue to explore the data provided statistically significant relationships are found.

It will be shown that this can result in many statistical tests, and can be contrasted with the traditional means of using hypothesis tests in which a

[4] Meaning those that are statistically significant.

single or a small number of predefined hypothesis tests are performed. The model also provides for a convenient means of presenting and summarising the many test results.

The model is demonstrated by application to the domain of learning and teaching of novice programmers using data that was described in Chapter 3. In this domain, the aim of the research is to investigate whether relationships exist between aspects of student behaviour, their learning style and their achievement. However, the selection of attributes must be done with regard to their relevance for finding answers to appropriate research questions; this will be examined in section 4.6.

4.3 Definition of terms

This section defines those terms that are required to explain the close ties between the data and the nature of the research model. The most important terms used in the method are object, attribute, child attribute, variable, and dimension.

Object: an entity that has attributes. In this application of the model, the object under examination is the student.

Attribute: a characteristic of an object that is measured. In the application of the model, the top level attributes of interest are learning style, achievement and behaviour, but many other child attributes are exposed in the research process. See also Variable.

Child attribute: a component of an attribute that has been aggregated with others into the attribute. Some attributes have child attributes; if present, the child attributes can be identified when the attribute is disaggregated. For example, the event log is a record of student activity (section 3.8) and the attribute *total events* is an aggregate of the number of events in each of the six P-Coder views (Figure 3-12), hence this attribute may be disaggregated into six child attributes.

Variable: definition is as for attribute above. However, in statistics, the term variable (independent and dependent) is inextricably linked with

hypothesis testing and so will be used instead of attribute in most statistical discussions.

Dimension: an n-dimensional (*n*-D) attribute is an aggregation of a number (*n*) of children and may be disaggregated into these components. A one dimensional (1-D) attribute cannot be disaggregated since it is already in its most primitive form.

Having defined these terms, it is now possible to examine the research model in more detail.

4.4 The model structure

The overall structure of the research model is illustrated in the context of the experimental domain in Figure 4-1 and its components are described in Table 4-2. This section explains the relationship between the *Learning Environment*, which is used as an experimental domain that produces the data, and the *Research Environment*, which uses the data as input to the research process.

4.4.1 The learning environment

The *Learning Environment*, Figure 4-1, represents the environment provided by an educational institution. Within this environment, the components are the actors, processes, tools and data stores. Each of the processes creates the data for its associated data store that will be the input to the *Research Environment*.

In the following paragraphs the actors are described and, subsequently, each of the processes and their associated data stores are explained.

The student

Students bring various past experiences and an assortment of current life situations to their studies. They have different expectations and motivations, yet they are presented with the same learning activities to develop and apply skills and knowledge. Student behaviours are uniquely individual, and they not only exhibit different behaviours, but also influence the learning and teaching process by those behaviours.

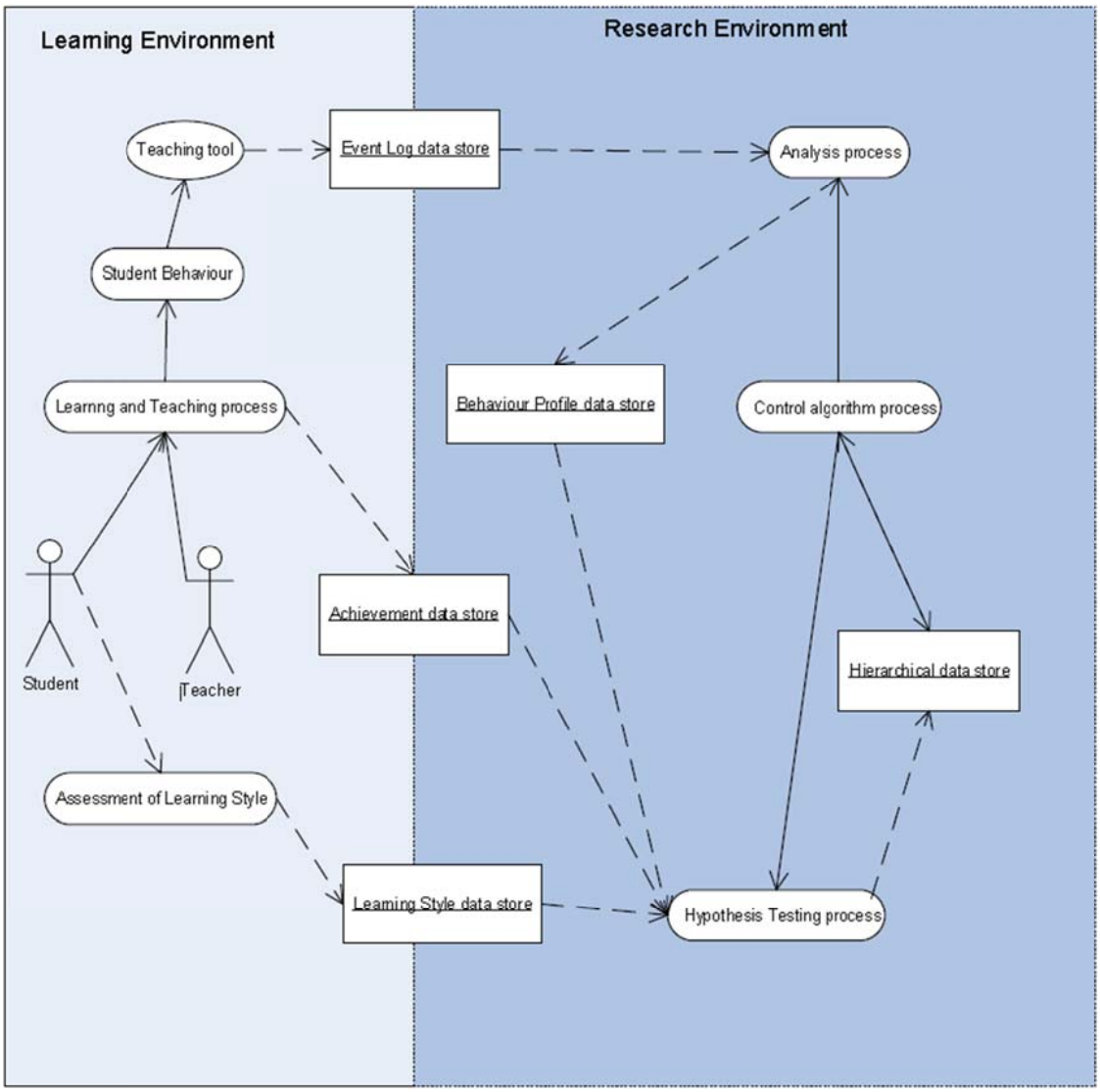
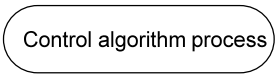
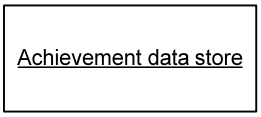
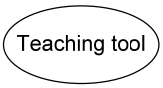


Figure 4-1. The structure of the model

The teacher

This is the person who both directs and delivers the curriculum through the learning and teaching process. The teacher is responsible for selecting the pedagogical processes with which the student is expected to engage and for setting and marking the exams that result in the measure of the achievement of the student. Although the teacher clearly has a significant influence on the *Learning Environment*, this research does not record any data on teachers directly but rather concentrates on the behaviour and learning outcomes of students. To simplify the model it is assumed that the effect of the teacher is consistent across all students; as a result this is not considered when analysing the behaviour of individual students.

Table 4-2. The components of the model

Diagram Element	Description	Example
Actor	A person that plays a role in this system	
Process	A sequence of actions	
Data store	A data set that provides a persistent record	
Tool	A software teaching tool used to record behaviour and collect data	
Relationship	A connection between processes	
Data flow	Data is moved either into or out of a data store	

Assessment of learning style

The assessment of learning style process is responsible for the creation of the data that is the input to the *Learning style data store*. There are several alternative learning style inventories (LSIs) that could have been appropriately applied; several of these were discussed in Chapter 2, and the process of assessing and selecting an LSI for this research was described in Chapter 3.

Learning style data store

The *Learning style data store* forms part of the interface between the *Learning Environment* and the *Research Environment*. This data is the input to the *Hypothesis Testing process* in the *Research Environment*. Learning style is considered to be constant for an individual in this research since the data is collected over the short period within a single semester. The issue of learning style changing over time was discussed in Chapter 2.

Learning and teaching process

The most important activity in the *Learning Environment* is the *Learning and Teaching process* in which both the student and teacher are engaged. The teacher is largely responsible for directing the process but it is also influenced by both the student and the environment. The model assumes

that the learning and teaching process will influence student behaviour. An essential outcome of this process is the measure of student achievement recorded by the teacher.

Achievement data store

The *Achievement data store* is a required product of the *Learning and Teaching process* and its contents represent a measure of each student's achievement during the semester. This was discussed in section 3.3.

Student behaviour

Students exhibit various behaviours, in the *Learning and Teaching process*. The behaviours of interest relate to the student learning experiences in using a *Teaching tool*. This behaviour is recorded via the tool in the *Event Log data store.*

Teaching tool

The *Teaching tool* is the software artefact that is used by the student in the learning process. This behaviour is recorded in the *Event Log data store* (section 3.6 explained the model of recording data and described the limitations of this process).

Event log data store

The *Event log data store* is an automated record of the events that provides evidence of some aspects of student behaviour as they use the *Teaching tool*. For a detailed description of the Event log used in this study, see section 3.6.

4.4.2 The research environment

The outputs from the *Learning Environment* are the three main data stores of *Learning Style*, *Achievement* and the *Event log*; these are inputs to the *Research Environment*. The other components are:

- Control algorithm process
- Analysis process
- Behaviour profile data store
- Hypothesis testing process
- Hierarchical data store

Each is described in the subsequent paragraphs.

Control Algorithm Process

The *Control Algorithm Process* (CAP) drives the research process and is at the core of the model. The process commences with two starting attributes, generates an initial hypothesis and presents the hypothesis for testing; it then records the result of the test, determines if there are new hypotheses to test and governs when to stop. A complete description of this process is given in the next section.

Analysis process

The *Analysis process* takes the raw data from the *Event log data store*, and, through a series of sorting, classification, filtering and aggregation techniques, configures the data into a *Behaviour profile* data store that is suitable for the input to the *Hypothesis testing* process.

The CAP indicates the hypothesis of interest from the *Hierarchical data store* and this is used to specify the particular *Behaviour profile* that is required. It may be necessary to perform reprocessing to extract data from the *Event log data store* and formulate the requisite *Behaviour profile*. For example, to test whether the total number of events affects achievements, all entries in the log for each student are summed; the student records are then sorted by the number of entries before hypothesis testing.

Behaviour profile data store

The *Behaviour profile* data store is the output from the *Analysis process*. It arises from the processing of data from the *Event log* data store in a format suitable for input to the *Hypothesis testing* process.

Hypothesis testing process

The *Hypothesis testing* process is a standard statistical process that applies appropriate tests to accept or reject a null hypothesis in the customary manner.

Hierarchical data store

The *Hierarchical data store* records the hypotheses to be tested and also the results of these tests. Since the data is explored in a hierarchical manner, this data store will naturally result in a tree like structure. Any statistically

significant relationships that are discovered in the data will result in a more detailed exploration of that data under the direction of the CAP.

4.5 Defining the Control algorithm process

The overall objective of the *Control Algorithm Process* (CAP) is to ensure that the data are systematically examined by hypothesis testing all of the relevant relationships that can be found in the attributes. The CAP identifies the hypotheses and manages them as they are presented for testing; it also describes how the hypothesis test results are recorded. The details of the CAP are introduced in two stages, a simplified version (Figure 4-2), followed by a more complete description of the algorithm (Figure 4-3).

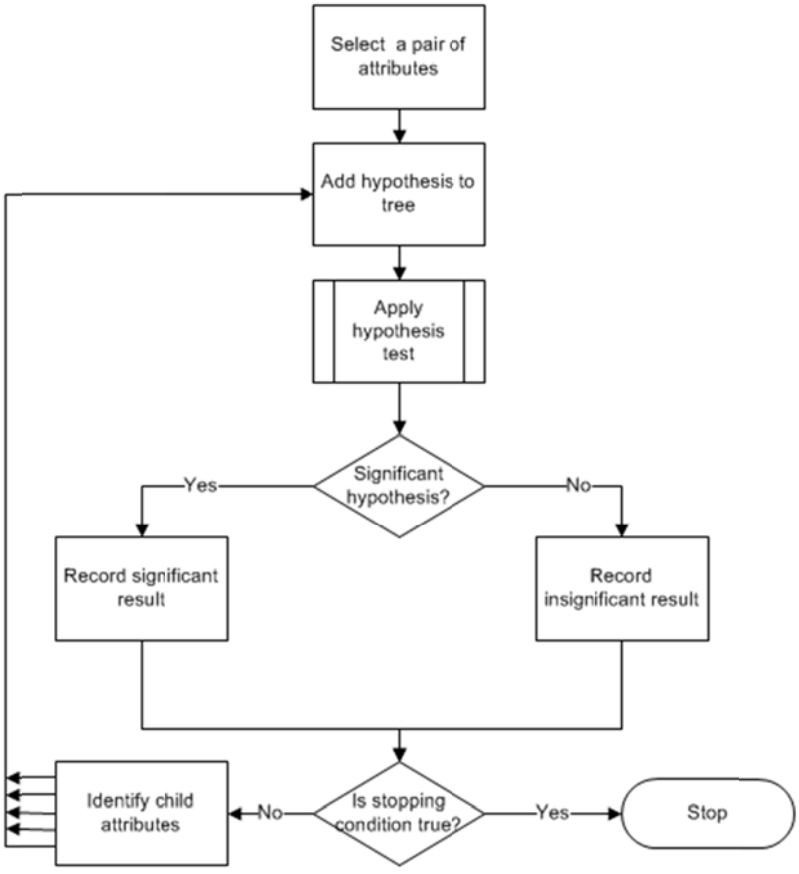


Figure 4-2. Overview of the CAP

The CAP must be initialised or seeded with a pair of attributes that will form the first hypothesis (Figure 4-2); they are selected to identify a relationship that might exist in the data. The attributes then become the

independent and dependent variables of the primary hypothesis. This initial hypothesis forms the root of a hypothesis tree and, following the required statistical analysis, the results are encoded into the tree. Then a decision as to whether to continue is made, the stopping condition is based on the results so far and the available data; this will be defined in more detail shortly. If the process is to continue, at least one of the current attributes must have child attributes. If so, then these are systematically formed into new hypotheses and added to the tree to begin the process again. Section 4.5.1 will explain in detail the manner that the child attributes are handled.

A detailed description of the CAP is provided in Figure 4-3.

The CAP has four sub-processes (defined at each  in Figure 4-3), *Initialise* where the process begins, the main *Process Hypothesis* method, *Disaggregate*, and *Check for Duplicates*.

Processing begins with *Initialise*, which is used only once, to form the root of the hypothesis tree from the appropriate attributes; the recursive *Process Hypothesis* sub-process is then called. It may be called many times during the application of the process. It begins by ensuring that the hypothesis is in a form that can be tested; that is the attributes are quantifiable and there is sufficient data. However, if one or both of the attributes cannot be measured then it will be impossible to perform any statistical test (discussed in section 5.1). If any hypothesis is not in a form that can be tested because either the independent or dependent variable has no available measure then it is deemed unquantifiable. In some cases it may not be possible to test the hypothesis, as there is insufficient data, in which case it is marked for no further testing.

If a hypothesis can be tested, then it is and the result is encoded into the hypothesis tree. Whether or not this test is possible, the algorithm continues to the next step, which seeks to investigate the data at a more detailed level.

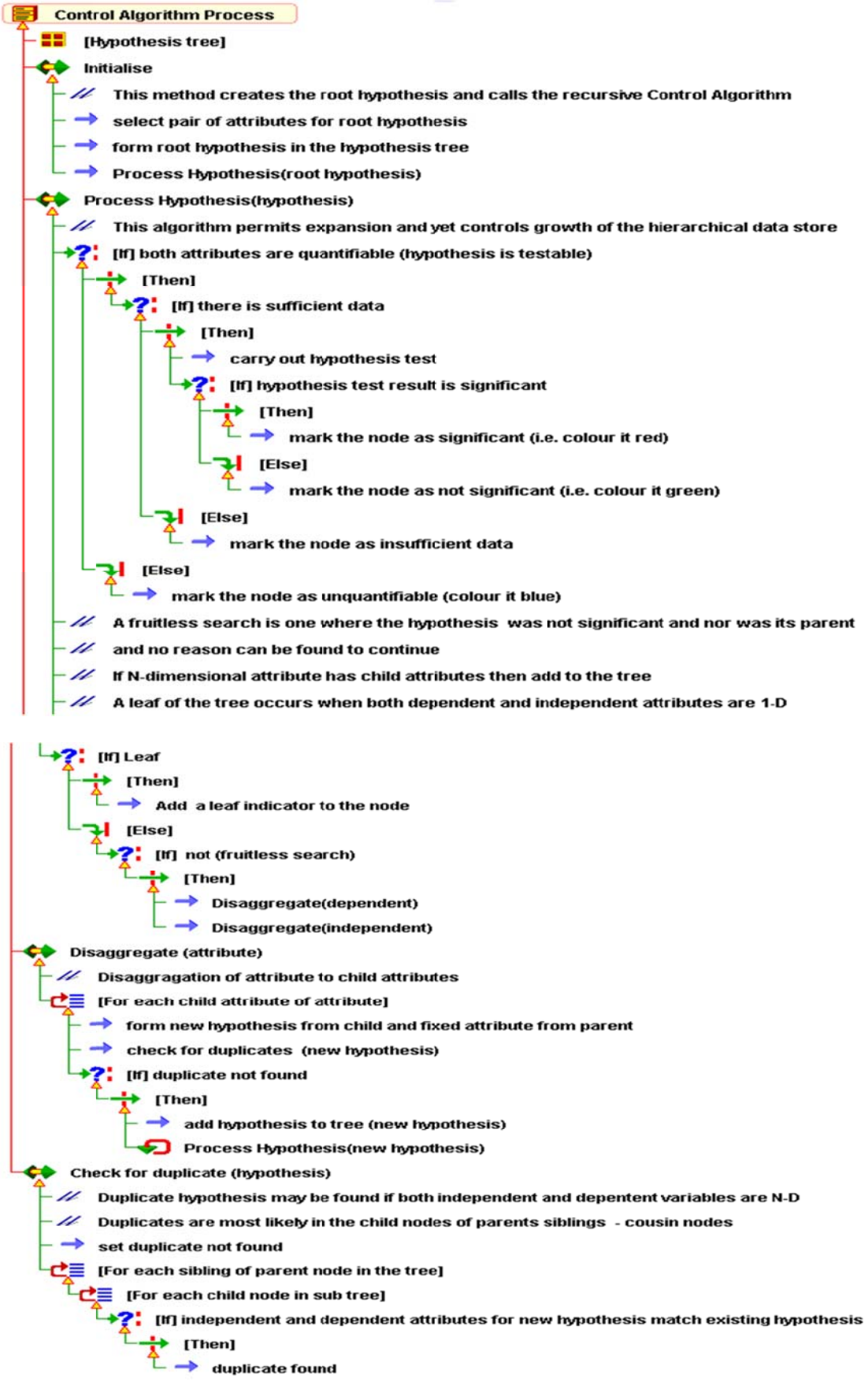


Figure 4-3. Detail of the CAP

4.5.1 Initialising and disaggregating

Initialising the CAP requires two attributes of the same object. The first step it to select the dependent and independent variables to formulate a hypothesis; such as:

H_0 : (Attribute A has no effect on attribute B)

H_1 : (Attribute A has an effect on attribute B)

A hypothesis pair of this type will be expressed as H (A=> B). Given measures for both attributes A and B, it is possible to formalise the hypothesis and place it at the root of the tree. As the CAP is underway and, following the testing of the initial hypothesis, disaggregation of the attributes is directed by the algorithm. If an attribute has n child attributes it is called *n*-Dimensional (*n*-D) and the process continues by forming hypotheses with the child attributes in the following manner.

If A is *n*-D and B is *m*-D then for each child node B_i of B, add the hypothesis H(A=>B_i) to the tree. Also at the current node, for each A_j of A, add the hypothesis nodes H(A_j => B). This process ensures that all possible relationships between each pair of the child nodes are considered (Figure 4-4). This illustrates all the combinations of relationships that may be of interest and also provides a structure that clarifies the relationships between the hypotheses themselves.

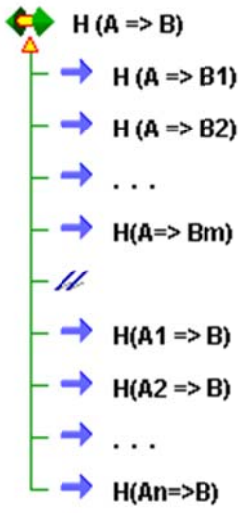


Figure 4-4. Child hypotheses are added to the hypothesis tree

4.5.2 Tree expansion

The hypothesis tree is expanded by disaggregating, when possible, the independent and dependent attributes by identifying their child attributes. Each child attribute is formed into one or more hypotheses and is placed in the tree according to the principles illustrated in Figure 4-4. The *Check for duplicates* process may be needed if both attributes are *n*-D, this ensures that newly formed child hypotheses are not already in the tree, if not, each is then submitted to the CAP in a recursive manner. The expansion of the tree through disaggregating attributes is only possible when the attributes in question are *n*-D; should they both be 1-D, then the limit of the current subtree has been reached.

Another reason for ceasing to expand the tree is that through repeated accepting of the null hypothesis, i.e. a particular branch is proving fruitless and no apparently sound argument can be found to examine it in greater depth. However, if it is suspected that aggregating the data may have hidden something interesting then it may be desirable to continue to search to lower levels in the tree. In the experimental application of the CAP for this study, the CAP was defined to continue to one more level when a hypothesis was found not to be significant. The consequence of this decision will be discussed in section 5.7.4, after the complete results have been presented.

The size of the resultant tree is dependent on two major factors; first, the number of dimensions that are uncovered that have data available and second, whether the hypothesis testing is providing any interesting results. If nothing of interest is discovered then the search may be identified as fruitless; this halts the expansion of the subtree and marks the node of the tree as pruned (see section 4.6)

The CAP defines the means by which the data are explored but also provides a way of controlling the potential exponential growth of the hypothesis tree. The tree is created by methodically and objectively expanding each node of the tree with the aim that all dimensions of the attributes of interest are tested.

4.5.3 Duplicate hypotheses

Since one of the objectives of the control algorithm is to prevent excessive growth of the hypothesis trees, it is important to avoid unnecessary repetition in the hypotheses that are formed. Duplicate hypotheses occur naturally in the case when both the independent and dependent attributes in a hypothesis are *n*-D and the tree is expanded to several levels. A theoretical example (Figure 4-5) will be used to illustrate this.

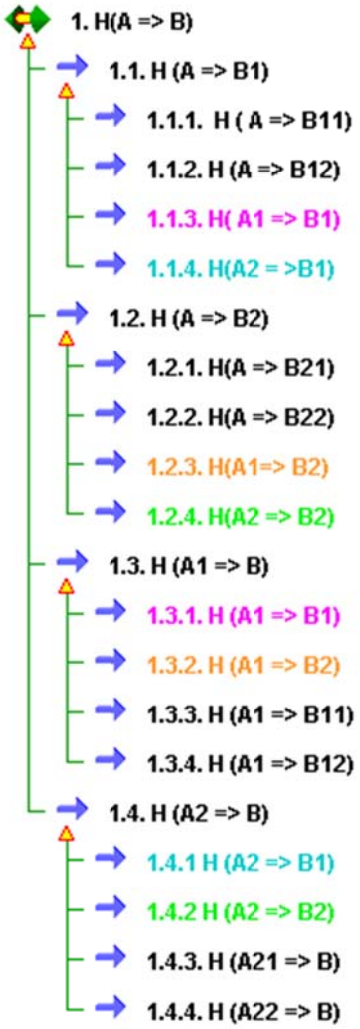


Figure 4-5 A theoretical hypothesis tree shows duplicates

Attributes A, B are formed into the seed hypothesis that is placed at the root of the tree. All attributes (A, B, A1, A2, B1 and B2) are 2-D, so as the tree is expanded, by disaggregating the attributes into child attributes and adding the hypothesis to the tree, there are four child hypotheses added to each node. These are hypotheses formed from each of the parent attributes paired with each of the child attributes of the other parent. Duplication of

hypothesis will arise at level 3 (and subsequent levels) in the tree if no prevention mechanisms are employed.

In Figure 4-5 duplicate nodes are indicated by matching colours[5]; there are four matching pairs of hypotheses (1.1.3 and 1.3.1), (1.1.4 and 1.4.1), (1.2.3 and 1.3.2), (1.2.4 and 1.4.2). The *Check for duplicates* method in the CAP will ensure that this kind of duplication does not occur by preventing the second hypothesis in each pair from being added to the tree. This is one way in which combinatorial explosion in the number of hypothesis can be controlled.

4.5.4 Related hypotheses

There are no duplicate hypotheses in the tree because the CAP directs that these are identified and discarded as the tree is created. However, there are numerous hypotheses that are related, in that, the data in either the dependent or independent variable is a subset or has a significant intersection with the dataset that used in another hypothesis. A child hypothesis is naturally related to its parent because of the process of disaggregating data. In some cases other hypotheses may also use intersecting datasets, For example, in a hypothesis tree that tests temporal data, it may be desirable to conduct both interval (e.g. minutes/days/weeks 1-3, 4-6, 7-9) and cumulative data (e.g. minutes/days/weeks 1-3, 1-5, 1-7) against a dependent variable. There is clearly a great deal of overlap in the data used in each of these cases. The issues associated with related hypotheses will be discussed after the results have been presented in section 5.7.4.

4.6 Presenting the results

As the control algorithm is applied to the hypothesis tree, the results of the hypothesis testing are encoded into the hypothesis tree, colouring its nodes to indicate its state. This is a simple and effective means of directing attention to those attributes that appear to be most interesting. The hypotheses in the tree are coded according to this scheme.

[5] The colours are added solely to demonstrate duplicates and not part of the encoding scheme

Unquantifiable – marked (U)

Blue : Indicates an unquantifiable node in the tree where the hypothesis cannot be tested because there is no available measure to test it. e.g. Hypothesis: The Amount of work affects achievement. This cannot be tested because amount of work has no available measure; the search can only continue by disaggregating the data into that of the child attributes, if any.

Not significant – marked (X)

Green: Indicates that hypothesis testing has been applied and the null hypothesis has been accepted so that the test is not particularly interesting

Significant – marked (S)

Red: Indicates that hypothesis testing has been applied and the null hypothesis has been rejected; this indicates a hypothesis of interest.

Insufficient data

Black: indicates that there is insufficient data to perform the hypothesis test.

Table 4-3. Legend for hypothesis tree

Symbol	Element	Comment
P affects Q(U)	Unquantifiable node	Nodes of this type cannot be tested
P affects Q (X)	Null hypothesis accepted (Non-significant result)	Two non-significant results at parent and child nodes will result in pruning
P affects Q (S)	Null hypothesis rejected (Significant result)	Significant results lead to disaggregating the data where possible.
P affects Q	Insufficient data	Not enough data available to perform the hypothesis test
!!!!	Tree pruned	No further disaggregation of data is done because stopping condition is reached
<<<<	A leaf node	No further disaggregation of data is possible because of the nature of the data. Coloured as for the node.

In addition to the colour coding, in some cases additional notation is added to a hypothesis node. A node is identified as a leaf of the tree when both attributes are 1-D and it will be marked accordingly (Table 4-3). This indicates that there are no further hypotheses in this subtree. This notation is

used regardless of whether the node was found to be significant or not. However, if the control algorithm directs that no further expansion of tree will occur, when both nodes are not 1-D, then the subtree is marked as pruned (Table 4-3).

Now that the CAP has been defined, the means of expanding the hypothesis tree has been explained and the manner of recording results is known, the following section will explain how this is applied to the experimental domain.

4.7 Formulating the root hypothesis in the experimental domain

The first step in applying the CAP in any domain is to select the two attributes that are suitable for formulating the primary hypothesis. These attributes will seed the algorithm and form the root of the tree.

In the experimental domain of this research, the purpose is to find out whether the learning style and/or the behaviour of students affected their achievement and also whether the learning style of students affected their behaviour. Figure 4-6 illustrates that there are potential relationships between any two of these datasets; however, it does not specify any direction of, or dependency between, these relationships.

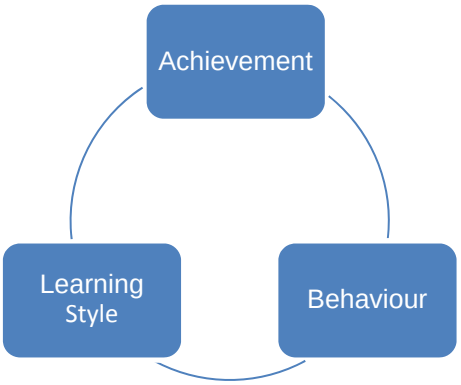


Figure 4-6. The three data sets in the study

In other research it may be reasonable to deem each of them to be either an independent or dependent variable, but the research questions posed in section 1.3 clarify the objective and demonstrate the associations between the attributes.

RQ2. Does learning styles affect achievement?
RQ3. Does behaviour affect achievement?
RQ4. Does learning styles affect behaviour?

These questions are now ready to be reformulated as hypotheses; however, the application of the control algorithm will soon require that the dimensionality of each of the attributes be known. Each of the questions in turn, especially in relation to the dimensionality of the attributes, will therefore be explored. The hypothesis tree is encoded and presented here to provide examples of the results. The statistical analysis will be explained in section 4.8 and the process of applying the algorithm to obtain the full results is described in Chapter 5.

4.7.1 Does learning style affect achievement?

This question places learning style and achievement as the independent and dependent variable, respectively. Since learning style is considered invariant (for the short duration of the study) and the purpose of the research is to seek to improve learning outcomes for students, this is the only reasonable way of connecting these attributes.

To find out whether there is a relationship between the preferred learning style of students and their achievement, the question is reformulated as a hypothesis (Figure 4-7).

H_0 : The preferred learning style of students does not affect achievement

H_1 : The preferred learning style of students does affect achievement

Figure 4-7. Hypothesis relating learning style and achievement

This hypothesis relates learning style to achievement; the first step in the operation of the CAP is to place the hypothesis at the root of the tree and test it. In this case the hypothesis test resulted in the null hypothesis being accepted and therefore coloured green (Figure 4-8).

The control algorithm then directs that child hypothesis are formed from any child attributes and that these are added to the tree. In this case the child attributes of the learning style are determined from the KLSI. They are the Perception (Concrete/Abstract) and Processing (Active/Reflective) continua (Figure 4-8). Throughout this study, a 1-D measure of achievement has been used and so it cannot be disaggregated. Hence two new hypotheses are formed and added to the tree, which is referred to as the Learning Style – Achievement tree (LS-A).

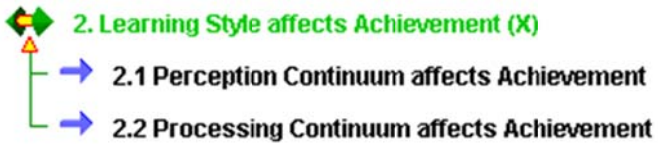


Figure 4-8. LS-A hypothesis tree including child hypotheses

Although small, this tree has reached its size limit because all of the attributes at the second level are 1-D. The second level nodes are therefore marked as leaf nodes (for notation see Table 4-3) and the resulting hypothesis tree is extremely shallow; hence any question of pruning the tree is irrelevant. The complete results of the hypothesis tests will be presented in Chapter 5 following a discussion of the statistical methods in section 4.8.

4.7.2 Does behaviour affect achievement?

This question places behaviour and achievement as the independent and dependant variables, respectively. Hence, the desirable hypotheses for testing are stated in Figure 4-9.

H_0 : Student behaviour does not affect achievement

H_1 : Student behaviour does affect achievement

Figure 4-9. Hypothesis relating behaviour and achievement

This hypothesis is placed at the root of the Behaviour-Achievement (B-A) hypothesis tree. The B-A root hypothesis is not tested because at this level Behaviour is an abstract concept and is therefore unquantifiable. However it is *n*-D and the number of dimensions is defined by the available data; this was described in Chapter 3. The CAP directs that the root is marked unquantifiable (for notation see Table 4-3) and the attributes are disaggregated. The child attributes are then formed into hypotheses and

added to the tree. In this study, metrics for three dimensions of Behaviour are available from the event log (section 3.8) and Behaviour is therefore a 3-D variable; these are measures of volume, time and effectiveness.

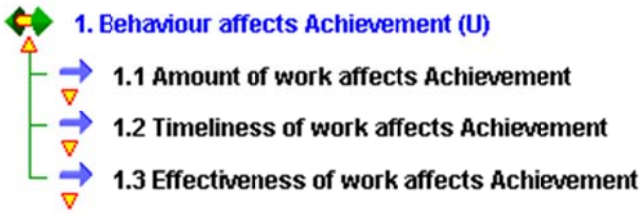


Figure 4-10. The root and first children in the B-A hypothesis tree.

As the CAP is applied to the B-A tree, it will be discovered that all three hypothesis at the second level are also unquantifiable. Hence, each will be encoded as for the root hypothesis and the three child attributes of behaviour will be disaggregated, new child hypotheses formed and the tree extended to level 3 (Figure 4-11).

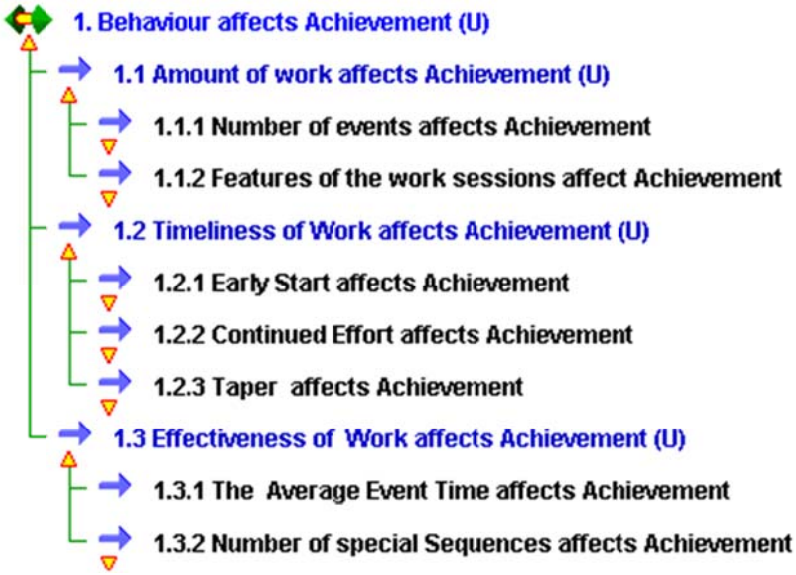


Figure 4-11. The B-A hypothesis tree expanded to two levels

At this level many of the attributes are quantifiable and so the hypothesis testing can begin. Since the complete process is extensive it will be presented in Chapter 5.

4.7.3 Does learning style affect behaviour?

In this question, learning style is the independent variable and behaviour the dependent one. The nature of the 3-D attribute, behaviour, was discussed in relation to the previous question; however here, behaviour is the dependent

variable. This question aims to locate those behaviours that are affected by learning style. The result of restating the question as a hypothesis is illustrated in Figure 4-12.

H_0 : Student behaviour is not affected by preferred learning styles.

H_1 : Student behaviour is affected by preferred learning styles.

Figure 4-12. Hypothesis relating learning style and behaviour

This hypothesis is placed at the root of the Learning Style - Behaviour tree (LS-B) in Figure 4-13. As for the B-A tree (section 4.7.2), behaviour is unquantifiable, and under the direction of the CAP, this is encoded into the tree. Then the attributes are disaggregated into child attributes, formed into hypotheses and added to the tree (Figure 4-13). Since the independent variable, learning style, is 2-D and the dependent variable, behaviour, is 3-D, there are five resulting child hypotheses to add to the tree.

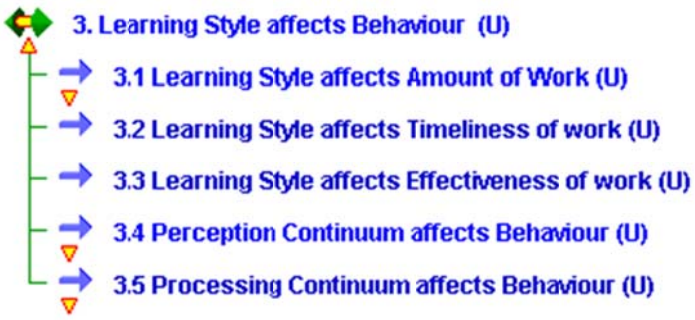


Figure 4-13. The root and first children in the LS-B hypothesis tree

All of these new hypotheses are also unquantifiable and will need to be disaggregated before any hypotheses in this tree can be tested. These results will be presented in the next chapter.

4.8 The statistics used in hypothesis testing

The discussion in this section applies to issues in the use of statistical tests in the particular experimental domain and to the particular datasets that were used in this trial of the model. Similar decisions would have to be made when using other datasets or in other domains, as the issues are the same, although it is possible that some specific outcomes may be different.

In this model all of the hypotheses test pairwise variables and so have a single independent and a single dependent variable. In the application of the model the student is the object under investigation so the discussion that follows uses this object rather than anything more general.

The independent variable in the B-A tree was behaviour, so to test the hypotheses in this tree, the students are placed in five approximately equal sized groups. These groups range from those with low levels of activity, through the mid ranges to high activity groups. The alternative would have been to define different ranges of activities and then assign individuals to these ranges. That grouping strategy could have resulted in a large variation in group size and hence some of the statistical tests would have been less robust.

A statistical test that is used to detect differences between groups is ANOVA (Analysis of Variance); more precisely, it detects differences in the means. It can be applied when the groups are independent, the data values are on a ratio scale, the dependent variable is approximately normally distributed, and variances are approximately equal.

The groups in all these tests are mutually exclusive. The data values of the dependent variables are on a ratio scale. The data are expected to be approximately normal but tests will be used to find out if this is true to an acceptable level. The Anderson-Darling and Shapiro-Wilk tests can be used to report evidence in the data that they are not drawn from a normal distribution (Sheard, Carbone, Lister, Simon, Thompson, & Whalley, 2008).

The result of an ANOVA test is an F value, the higher the F-value, the smaller the likelihood that any difference in the means of the groups arose by chance. The p-value, or probability of the F-value occurring by chance, is used as a determinant. If the p value is less than the predefined α then the null hypothesis is rejected and the result is defined as being statistical significant. Common values used (for α) are 0.05 and 0.01 (Sheard et al., 2008). In their studies Byrne and Lyon (2001) used 0.01 and Ventura and Rasmurthy (2004) used 0.05. The selection of an α value should take into consideration the interplay between Type I and Type II errors.

A Type I error occurs when the null hypothesis is rejected but it should have been accepted (Table 4-4). The probability of a Type I error is denoted by α. A Type II error, denoted by β, occurs when the null hypothesis is accepted but it should have been rejected. The precise value of β, is calculable and the probability of each of these types of error occurring is inversely related for any given sample size. So trying to control for a Type I error by selecting an extremely low value will increase the probability of a Type II error. If it is considered necessary to set the probability of both Type I and Type II errors, then the sample size must be selected appropriately.

In the experimental application of the research model in this research, a significant factor was that the size of sample was fixed (discussed in Chapter 3) and so decreasing α would increase the likelihood of Type II errors. Balancing these considerations, an $\alpha = 0.05$ was used. This research was looking for interesting relationships in the data, and in this type of exploratory research, making a Type I error does not have dire consequences. In fact, making an error of this kind may be considered preferable to not finding a relationship when one truly exists.

Table 4-4. Type I and Type II errors

	Retain Hypothesis	Reject Hypothesis
Null hypothesis is true	Correct decision	**Type I error : α** Incorrectly reject null hypothesis
Alternate hypothesis is true	**Type II error : β** Incorrectly retain null hypothesis	Correct decision

The implication of making a Type I error in this research is that the CAP will direct further hypotheses to be explored deeper into a subtree, when it may otherwise have terminated. It is possible that this would be visible in the tree by a significant hypothesis having no significant hypotheses in its children. It will be shown (in the summary of the trees at the end of Chapter 5) that this manifestation of a Type I error did not occur.

If Type II errors occur then the result could be that potentially interesting hypotheses will not be discovered because the search is prematurely terminated. An error of this kind may remain undiscovered but the decision,

to not terminate at the first non-significant hypothesis in a subtree but to continue to one further level, will mitigate against this risk. It was impractical to set β in this research because of the issues already discussed in Chapter 3.

Post-hoc testing is carried out when a significant difference is found between groups to identify the specific groups that are significantly different. When the groups are of equal size, Scheffé's test is appropriate. It was selected because it is relatively conservative and controls for the multiple tests that are done when looking for differences between any of the groups (Rountree, Rountree, Robins, & Hannah, 2004).

4.9 Limitations of the model

In this chapter a research model has been described that defines a process for systematically investigating relationships in the data using hypothesis testing. The model can be applied to two variables that are formed into a primary hypothesis.

This model describes a means of studying the relationships between pairwise attributes of an object. The model provides for a complete investigation into all relevant relationships in the data by disaggregation of n-dimensional attributes and the creation of a hierarchical data store.

This model could be applied to any investigation into any attributes of interest; however, in this investigation the model has been applied to three critical datasets of achievement, behaviour and learning styles. There are several limitations of this research; these can be categorised as limitations with the model and limitations with the application of the model. The former are discussed here and the latter in section 5.8.

The use of the CAP could potentially exclude some important questions if some child attributes are overlooked (an omission by the researcher). This is a human task and not an automated one and so it is the responsibility of the researcher to select child attributes appropriately.

Within the CAP the number of levels to continue beyond the first non-significant hypothesis must be selected. In this research this was set to two

levels, so that testing continued until two levels of non-significant hypothesis were found. There is some evidence in the results that this decision was reasonable, since a few significant hypotheses were uncovered by continuing to the second level. If more than three levels had been selected there would have been a much larger number of hypotheses required.

The model is designed for pairwise attributes. Although it is clearly possible to form hypothesis with multiple independent and/or dependent variables, this was not considered appropriate in this model because of the resulting complexity of the disaggregation process and the potential for combinatorial explosion.

4.10 Summary

This chapter has presented a research model that was designed to thoroughly and systematically investigate empirical data. The major contributions of this thesis are the model and its components, the control algorithm process, which manages the research process, and the hypothesis tree, which is used to record results.

In order to apply this model, it is seeded by a hypothesis that connects two variables of interest and it dictates the testing of a series of hypotheses; as it does this, it also delves deeper into the data to identify additional relationships.

The manner that the model can be applied to the domain of novice programmers has been described and some of the limitations of the model have been exposed. In the following chapter, the application of the model will be completed.

CHAPTER 5 DATA ANALYSIS: APPLYING THE MODEL

This chapter describes the application of the control algorithm introduced in Chapter 4 (see Figure 4.4). It is applied to each of the three primary hypotheses with the objective of completely investigating all interesting relationships in the data. Cases presented have been selected to illustrate the application of the CAP. The six cases are:

1. Initialising: a hypothesis at the root node
2. A standard hypothesis test with a significant result
3. A standard hypothesis test with a non-significant result
4. Halting due to no child attributes
5. Halting due to a fruitless search
6. Halting due to insufficient data

These cases have been selected to demonstrate the standard operations involved (1, 2, 3, and 6) and also to explain the issues involved when guidance from the researcher is required (4 and 5).

The operation of the model will be portrayed by the cases that illustrate the research process. The complete results are contained in hypothesis trees that are presented and summarised in section 5.7. The notation that is used in the hypothesis trees is defined in Table 5-1.

Operational issues that arose in the process of applying the control algorithm will be considered in section 5.8, while analysis and discussion of the results are in Chapter 6.

5.1 Initialising: a hypothesis at the root node

The application of the control algorithm requires a primary hypothesis as the starting point. Three such hypotheses were selected for this research, with each one formed from a research question. The first is transferred directly from Figure 4.7, “Student behaviour affects achievement” and is placed at the root of hypothesis tree (Figure 5-1). This tree will be referred to as the Behaviour-Achievement (B-A) hypothesis tree.

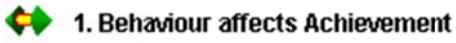


Figure 5-1. The root of the B-A hypothesis tree

The control algorithm can then be applied using the root hypothesis as a seed to initialise the process. The hypothesis itself is reproduced in Figure 5-2. The behaviour that is under consideration is the programming behaviour that has been captured and discussed in the previous two chapters.

Table 5-1. A legend for the hypothesis trees

Symbol	Element	Comment
P affects Q(U)	Unquantifiable node	Nodes of this type cannot be tested
P affects Q (X)	Null hypothesis accepted (Non-significant result)	Non-significant results at parent and child nodes will result in pruning
P affects Q (S)	Null hypothesis rejected (Significant result)	Significant results lead to disaggregating the data where possible.
P affects Q	Insufficient data	Not enough data available to perform the hypothesis test
!!!!	Tree pruned	No further disaggregation of data is done because of non-significant results.
<<<<	A leaf node	No further disaggregation of data is possible because of the nature of the data. Coloured as for the node.
x/y	Significance ratio.	This ratio is not simplified. It shows the number of significant hypotheses to the total number of hypotheses below (and including) this node in the tree.
n%	Density percentage	This is the significance ratio expressed as a percentage.

It should be noted that the application of the algorithm is not totally automatic. It requires the application of domain knowledge at several decision points, which are illustrated in the following sections.

H_0: Student behaviour does not affect their achievement.

H_1: Student behaviour does affect their achievement

Figure 5-2. The null and alternate hypotheses at the root of the B-A tree

In this study there is no available metric for student behaviour for which data have been collected, hence the attribute behaviour is unquantifiable and this hypothesis (Figure 5-2) cannot be directly tested. Accordingly the control algorithm directs that this hypothesis is marked as unquantifiable (U) and coloured blue, each of the attributes that is *n*-D is then disaggregated into its child attributes and new hypothesis formed based on each of these.

To ensure the most useful selection of child attributes, guidance is required to select attributes for which there are data available. In this instance, there are three child attributes to behaviour available in the data. These are the volume, temporal aspect and effectiveness of work. Hence, amount of work, timeliness of work and effectiveness of work were selected as the child attributes to be used. Since a one-dimensional measure of achievement (section 4.4.1) is used in this study, it has no child attributes. Hence there are three newly formed hypotheses that are added to the tree (Figure 5-3).

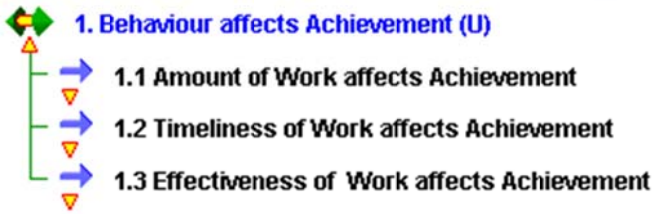


Figure 5-3. B-A hypothesis tree with new hypotheses formed from child attributes

Once more, each of these attributes is identified as having no available measure and hence being unquantifiable. The hypotheses are marked, the attributes disaggregated and new hypotheses formed based on the child attributes and these are added to the hierarchy (Figure 5-4). The first hypothesis that can be tested is *1.1.1 Number of Events affects Achievement.*

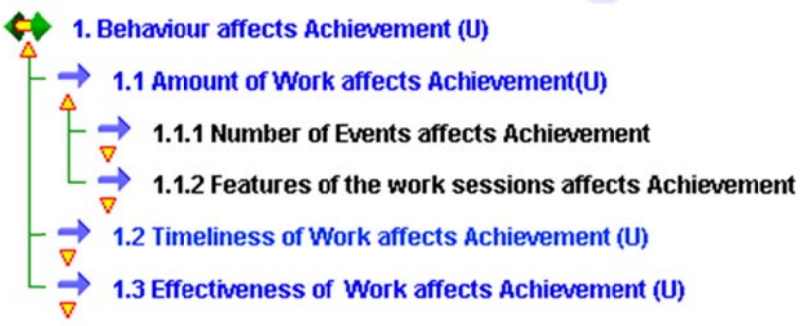


Figure 5-4. B-A hypothesis tree: A second expansion from child attributes

5.2 A standard hypothesis test with a significant result

At level 1.1.1 in the B-A tree (Figure 5-4), the hypothesis under consideration is created to answer the question, "Does the number of events affect the achievement of students?"

In this hypothesis the independent variable is a count of the number of events performed by students as they learn to program and provides an indication of the volume of work that they have completed in the duration of the study.

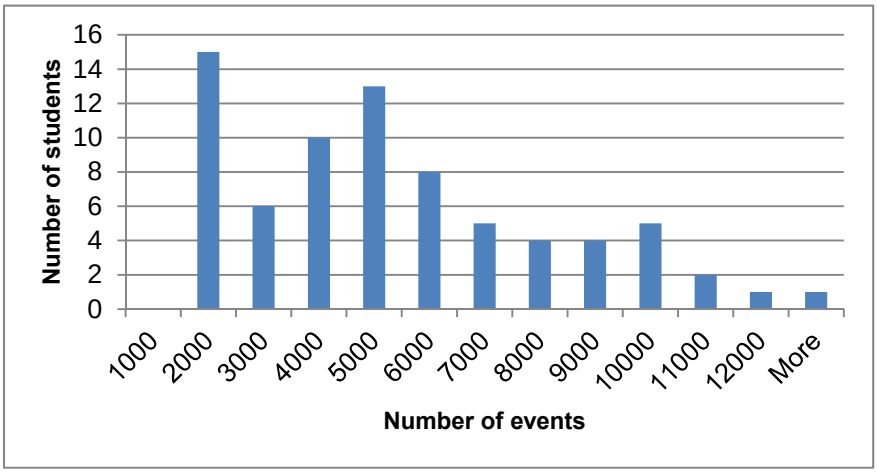


Figure 5-5. Frequency distribution of the total number of events

The frequency histogram (Figure 5-5) shows the variation in the number of events generated by students in the course of one semester. The mean number of events in this group was 4890 (s.d.2820). This distribution is multi-modal with a large number of students with a small number of events.

It should be noted that a number of students who completed less than 1000 events in the university computing laboratories were excluded from this analysis. This cut off level for number of events was selected in combination with two other factors, the number of episodes of working in the lab and the spread of these episodes through the semester. All students with 1000 or more events also worked in the lab at least five times and these events were spread across the semester. Below this level, students had very few separate episodes when they worked in the computer lab and those few, were at the very beginning of semester. Some were students who withdrew from the course but there were also some students who appeared to have

some experience at programming (given their explorations in the early weeks) and presumably chose to work on their home computers after that.

A scatter plot of the total number of operations against achievement levels does not clearly show any relationship between the two variables (Figure 5-6); the correlation coefficient between them is .30. There is a cluster of very low exam scores with a few operations and almost no scores below fifty for students with over 8000 operations for the duration of the semester.

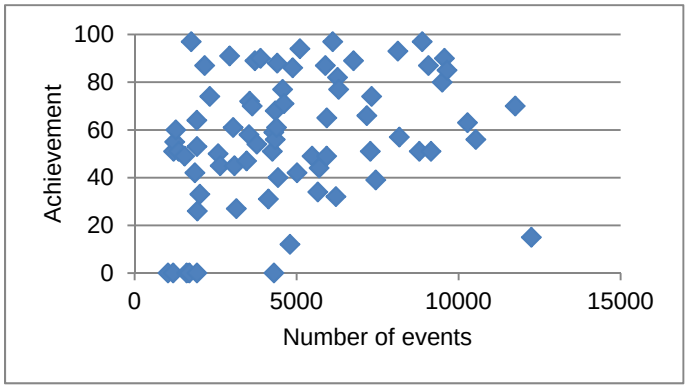


Figure 5-6. The number of events against achievement

To investigate whether there is a difference in the achievement of students given different activity levels, the null hypothesis and alternate hypothesis are formed (Figure 5-7). An analysis of variance is used to report on how likely it is that the groups are homogeneous.

H_0: Student achievement in five equal sized groups from low event activity levels to high event activity levels is the same.

H_1: Student achievement in five equal sized groups from low event activity levels to high event activity levels is different

Figure 5-7. Hypotheses connecting achievement and event activity

A one-way ANOVA tests for a difference in population means but assumes that the data are from populations that are approximately normal and have equal variance. However, ANOVA is known to be robust in respect to non-normality (McDonald, 2009). The Anderson-Darling A^2 test and the Shapiro-Wilk W test provide measures (A^2 and W) that indicate whether it is reasonable to assume that a dataset is normally distributed. When the probability (p) of the calculated value is low then the assumption of normality is likely to be incorrect. These tests were applied to the five

activity level groups. The results indicated that this assumption of normality was likely to be incorrect in the low activity group but correct in the remaining four groups (Table 5-2). However, given the robustness of the test, it was decided to continue with ANOVA rather than use the equivalent non-parametric test.

Table 5-2. The results of tests for 'non-normality' in activity level groups

	Low activity level	Mid-low activity level	Mid activity Level	Mid-high activity level	High activity level
Anderson-Darling A^2	0.86	0.25	0.26	0.34	0.32
p of A^2	0.020	0.712	0.663	0.459	0.485
Shapiro-Wilk W	0.87	0.96	0.95	0.94	0.94
p of W	0.034	0.632	0.484	0.358	0.413

A one-way ANOVA was used to test for differences in achievement among five activity level groups (Table 5-3). The F statistic is a ratio that compares the actual variation in the group means from the expected variation. An F statistic close to 1 would indicate no significant difference in the means of the groups and hence acceptance of the null hypothesis; whereas high values of F, suggest rejection. The determinant is the respective probability (p) of the resultant F.

Table 5-3. ANOVA result : Total events affects achievement

Groups	**n**	**Mean**	**SE**	**Pooled SE**	**SD**
Low	15	36.5	7.86	6.41	30.4
Mid-Low	15	60.2	5.14	6.41	19.9
Mid	15	55.5	6.93	6.41	26.9
Mid-High	15	66.0	5.62	6.41	21.8
High	14	66.7	6.31	6.63	23.6
Source of variation	**Sum squares**	**DF**	**Mean square**	**F statistic**	**p**
Groups	9006.6	4	2251.7	3.66	0.0092
Residual	42470.7	69	615.5		
Total	51477.4	73			

In this case, the high F value and low probability ($F(4,69) = 3.66$, $p=.0092$) indicate that achievement differed significantly across the five groups,. So the null hypothesis, Student achievement in five equal sized groups from low event activity levels to high event activity levels is the same, is rejected and the conclusion reached that it is extremely unlikely that the groups are homogeneous.

To identify which pairs of groups are significantly different, Scheffé's test can be used (Table 5-4. Scheffé's post-hoc test on total event activity levels and resultsTable 5-4). In this instance the Low activity level group is found to have significantly different results from both the Mid-High and the High activity level groups.

Table 5-4. Scheffé's post-hoc test on total event activity levels and results

Scheffé Contrast	Difference	95% CI	
Low v Mid-Low	-23.7	-52.3	to 5.0
Low v Mid	-18.9	-47.6	to 9.7
Low v Mid-High*	-29.5	-58.1	to -0.8
Low v High*	-30.2	-59.4	to -1.0
Mid-Low v Mid	4.7	-23.9	to 33.4
Mid-Low v Mid-High	-5.8	-34.5	to 22.9
Mid-Low v High	-6.5	-35.7	to 22.7
Mid v Mid-High	-10.5	-39.2	to 18.1
Mid v High	-11.2	-40.4	to 17.9
Mid-High v High	-0.7	-29.9	to 28.5

*significant

Figure 5-8 shows both the parametric diamond and the non-parametric 'candlestick' or central tendency box-plots of the five activity groups. The box plot allows straightforward comparison of several data sets (Hoaglin, Mosteller, & Tukey, 1983). The parametric diamond shows the mean and the 95% confidence interval for the mean. The format used for the box plots is that of outlier box plots that have whiskers extending to the furthest observations within ±1.5 IQR (interquartile ranges) of the 1st or 3rd quartile. The candle's mid line is the median and the candle extends from the 1st to the 3rd quartile. This graph illustrates the statistical findings and provides an

alternative but meaningful depiction of the data. Hoaglin et al.(1983) consider the median as a more robust measure of central tendency than the mean because it is less influenced by changes in the value of outliers.

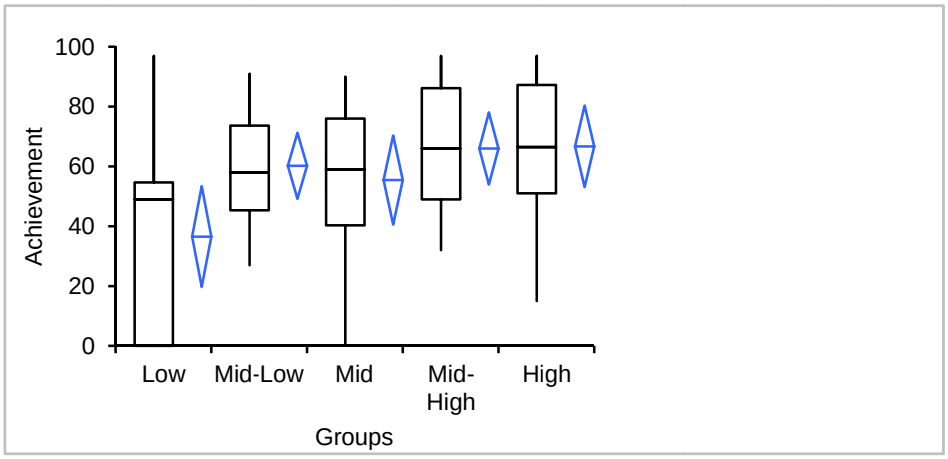


Figure 5-8. The achievement of students in five activity levels of all events

The mean of each group is at the centre of the diamond and the length of the diamond indicates a 95% confidence interval for the mean. Figure 5-8 indicates that students in the higher activity level groups are more likely to have higher achievement. However it also demonstrates that once above a low activity level, the difference in achievement by the groups is not as marked. The median of the lowest group is just below 50% which signifies that more than 50% of these students failed. The medians and means of the two highest activity level groups are close to 70% which makes evident the good achievement in these groups.

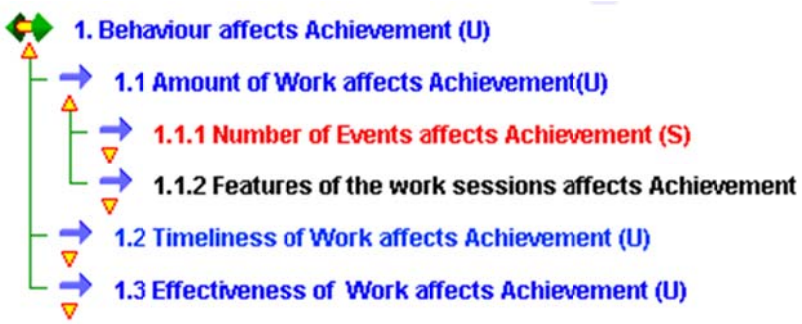


Figure 5-9. A significant result is encoded in to the B-A tree (Node 1.1.1)

Since a significant difference between the results of student groups with a different number of total events during the semester has been discovered, the appropriate node (1.1.1) in the hypothesis tree can be updated (Figure 5-9). The control algorithm then directs that the attributes that are *n*-D

should be disaggregated, child hypotheses formed and added to the tree (as described in section 4.4.2.1 Hypothesis tree and secondary testing); this is illustrated in Figure 5-10.

In selecting the child attributes that will be used to form new hypotheses, there may be decisions to be made about the criteria that will be used for subdividing the population. In other words, a judgment must be made about how they should be grouped. In this case, the child attributes that were selected were directed by the programming environment, P-Coder (Chapter 3). The five views, Design, Code, Object, Module and Class events correspond to different tasks in the software development process. They are defined by the development environment that students were using for programming and each represents a different kind of interaction. For example, Design events include adding lines of pseudocode to the program to design the program, and Code events include compiling and executing the program. It is possible that the investigation could have also included grouping pairs of these collections of events either instead of or in addition to the five selected.

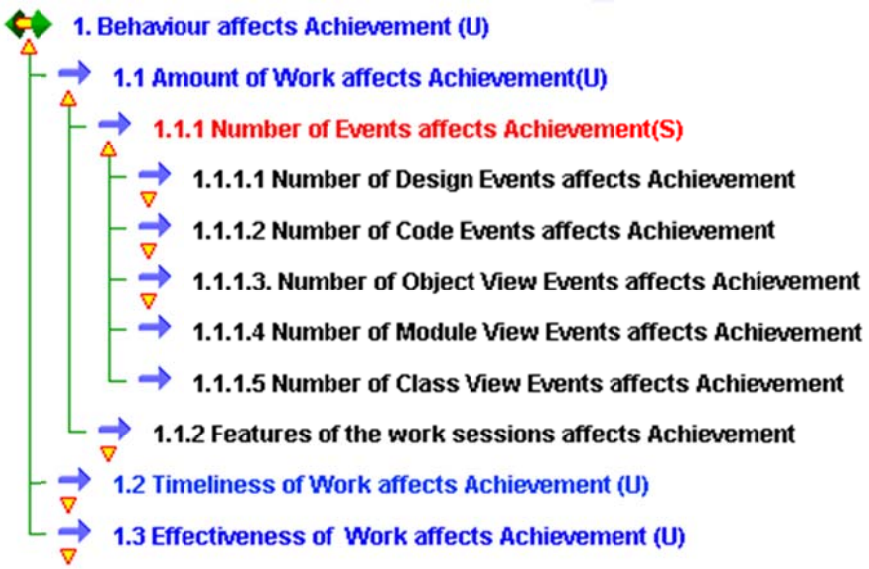


Figure 5-10. The child attributes form additional hypotheses in the B-A tree

Following the addition of each of the new hypotheses to the tree (Figure 5-10), the control algorithm directs the process to continue with each of these new hypotheses in turn. The process will eventually terminate when either leaves are reached or fruitless searches are identified at each node.

5.3 A standard test with a non-significant result

When investigating whether features of the work session affect achievement at 1.1.2 in the B-A hypothesis tree (Figure 5-10), "features of the work session" is unquantifiable. A more precise definition of the particular features in question is required. The child attributes for which data are available in this study are the particular features:

- Total time
- Length of sessions
- Number of sessions
- Regularity of sessions

At this point in the research the hypothesis tree is illustrated in Figure 5-11. It should be noted that, in this figure, the subtrees of 1.1.1 are not visible because they have been 'rolled-up' to reduce the size of the tree for viewing purposes.

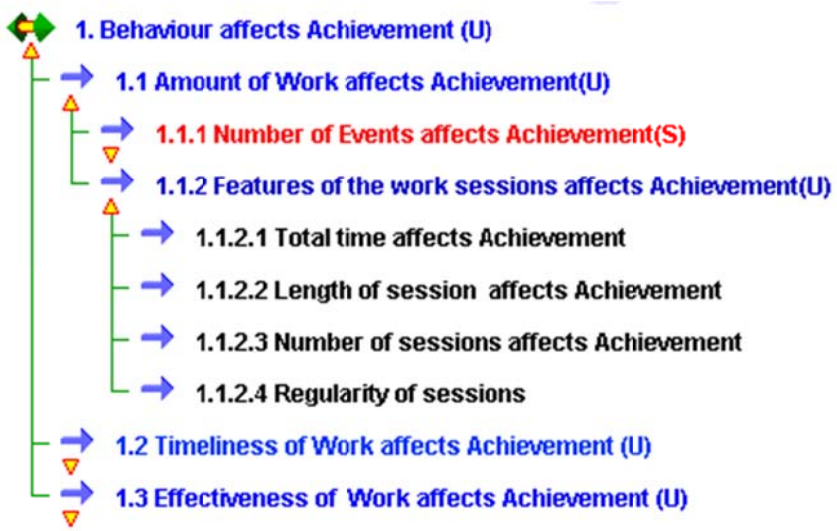


Figure 5-11. The child attributes of 1.1.2 are added to the B-A tree

The first of the child attributes, total time, is defined as the total amount of time during the entire semester that a student spends using P-Coder in the university computer labs. The histogram (Figure 5-12) shows the distribution of students over the number of hours spent programming in the university computer labs. There are a small number of students with a very large number of hours. The mean number of hours is 22.0 (s.d.11.4). The correlation coefficient between time spent programming and achievement is

0.29. This relationship is further illustrated in the scatter plot (Figure 5-13) and this does not show any association between the variables.

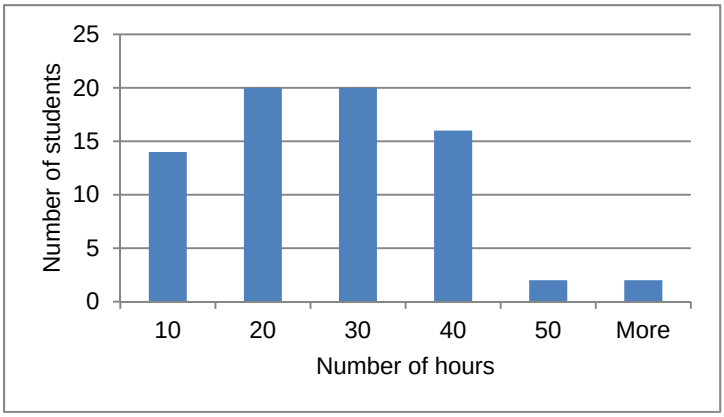


Figure 5-12. Frequency distribution of time spent programming

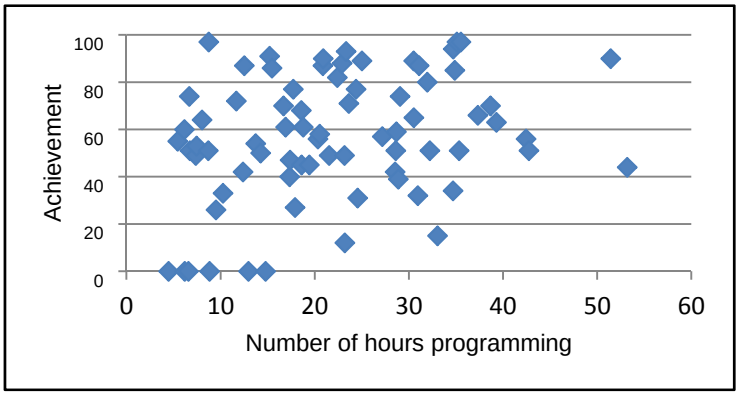


Figure 5-13. Scatterplot of time programming against achievement

The hypothesis to be tested, as before, is based on five groups from few hours to many and is formalised in Figure 5-14.

H_0 : The achievement of students no matter how much time they spend programming will be the same

H_1: The achievement of students in five equal sized groups from few hours to many hours will be different

Figure 5-14. The null and alternate hypotheses 1.1.2.1

To find out whether an ANOVA can be used, tests are applied to the populations to find out if there is evidence that they are not normally distributed. The results in Table 5-5 show that there is no evidence of this and so the hypothesis can be tested using the parametric test.

Table 5-5. Tests for non-normality in time-spent populations

	Low activity level	Mid-Low activity level	Mid activity Level	Mid-high activity level	High activity level
Anderson-Darling A^2	0.61	0.37	0.37	0.33	0.29
p of A^2	0.090	0.385	0.385	0.484	0.571
Shapiro-Wilk W	0.91	0.93	0.93	0.93	0.94
p of W	0.119	0.270	0.279	0.266	0.466

The result of the ANOVA test on the five groups is an F value of 2.27 (Table 5-6). The probability of this result, assuming the null hypothesis, is 0.07. At the 0.05% level, the null hypothesis cannot be rejected and the conclusion must be reached that there is no evidence that the groups are not homogeneous.

Table 5-6. ANOVA result: Time affects Achievement

Groups	n	Mean	SE	Pooled SE	SD
Low	15	40.9	7.79	6.63	30.2
Low-mid	15	53.6	7.43	6.63	28.8
Mid	15	63.6	5.77	6.63	22.4
High-mid	15	61.5	5.28	6.63	20.4
High	14	65.2	6.76	6.86	25.3
Source of variation	**Sum squares**	**DF**	**Mean square**	**F statistic**	**p**
Groups	5982.3	4	1495.6	2.27	0.0706
Residual	45495.0	69	659.3		
Total	51477.4	73			

Figure 5-15 illustrates the large variability in the results of students in each group; this is an indicator for a non-significant result. The non-significant result of the hypothesis test is then encoded into the tree (1.1.2.1 in Figure 5-16) and the control algorithm then directs that a decision is made whether or not to continue.

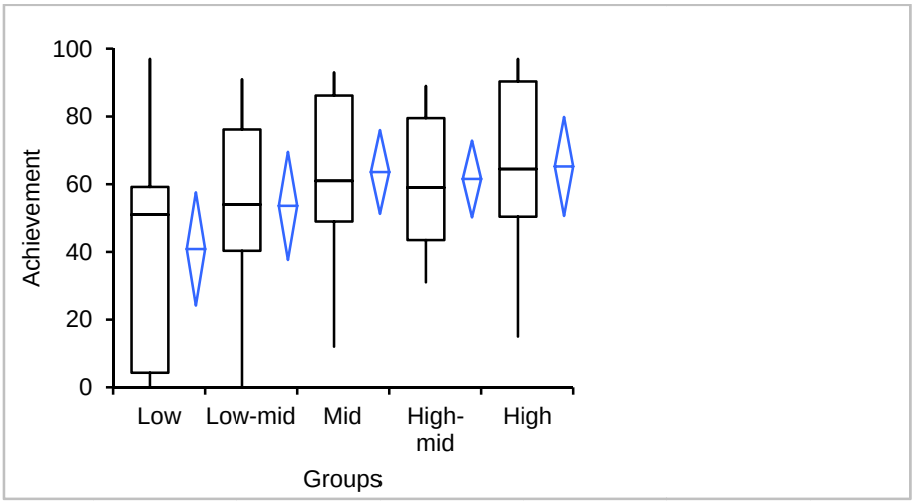


Figure 5-15. Mean and 95% CI of mean achievement for time spent programming

Since both total time and achievement are 1-D, there are no child attributes and hence this hypothesis is a leaf of the tree.

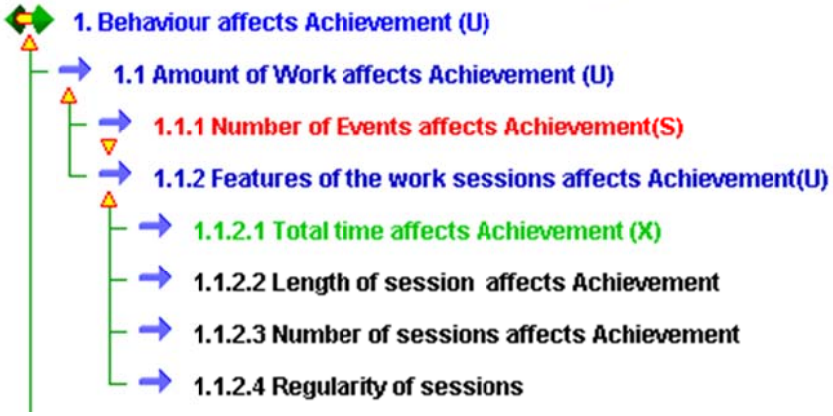


Figure 5-16. A non-significant result is encoded into the B-A tree at 1.1.2.1

5.4 Halting due to no child attributes

The second primary hypothesis in this research, tested if learning style affected achievement. This hypothesis was placed at the root of the LS-A tree (Figure 5-17) and was found to be statistically non-significant. The control algorithm directs the search to continue by identifying the child attributes of learning style and achievement. The measure of achievement used in this research is 1-D and so does not contribute to any further expansion of the tree. However, the measure of learning style used in this research is KLSI which is 2-D, with the perception and processing continua underlying the selection of the initial 4-way classification of learning style.

These child attributes of the independent variable can then be formed into hypotheses and added to the tree. Hence the hypothesis tree in Figure 5-17 has two new hypotheses that are ready for testing.

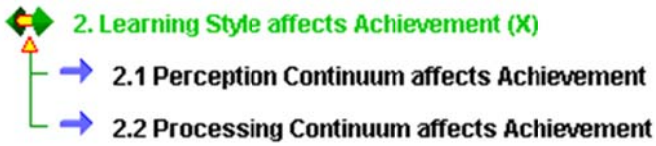


Figure 5-17. The LS-A hypothesis tree

To test along the Perception continuum using the available data, the two KLSI groups at opposite ends of this continuum were combined. That is, the Accommodating and Diverging groups were combined to form a Concrete group, while the Assimilating and Converging groups were combined an Abstract group. The resulting hypotheses are listed in Figure 5-18.

H_0: Student achievement in the Concrete and Abstract groups is the same.

H_1: Student achievement in the Concrete and Abstract groups is different

Figure 5-18. The null and alternate hypotheses 2.1

Since there are only two groups involved, a t-test is the appropriate statistic whereas ANOVA was used for multiple groups. The two-tailed t-test (Table 5-7) shows that the probability of achieving these results given the null hypothesis is 0.02. Since this is less than t-critical (0.05), the null hypothesis should be rejected and the conclusion reached that the achievement of students in the Concrete and Abstract Kolb dimension is different.

The difference between the two groups is clearly illustrated with the means of the two groups and 95% confidence intervals for the means (Figure 5-19).

Under the direction of the control algorithm the significant result of the hypothesis at 2.1 could be followed by the addition of further hypothesis to the tree. However, one situation that halts the control algorithm is that both independent and dependent variables are 1-D. If both variables are 1-D, then there are no child attributes and so the search process on this branch of the hypothesis tree will cease. This applies to node 2.1 of the hypothesis tree in Figure 5-20. The attributes under consideration here are the Perception continuum of KLSI and achievement and these attributes have no child

attributes. Hence, this node (2.1) of the hypothesis tree is identified as a leaf node and the control algorithm is directed to continue with the next branch of the tree (2.2).

Table 5-7. T-test for hypothesis 2.1

Groups	n	Mean	SE	SD
Abstract	30	60.3	4.21	23.1
Concrete	9	39.4	7.01	21.0
Mean difference	20.9			
95% CI	3.4 to 38.3			
SE	8.61			
t statistic	2.43			
DF	37.0			
2-tailed p	0.02			

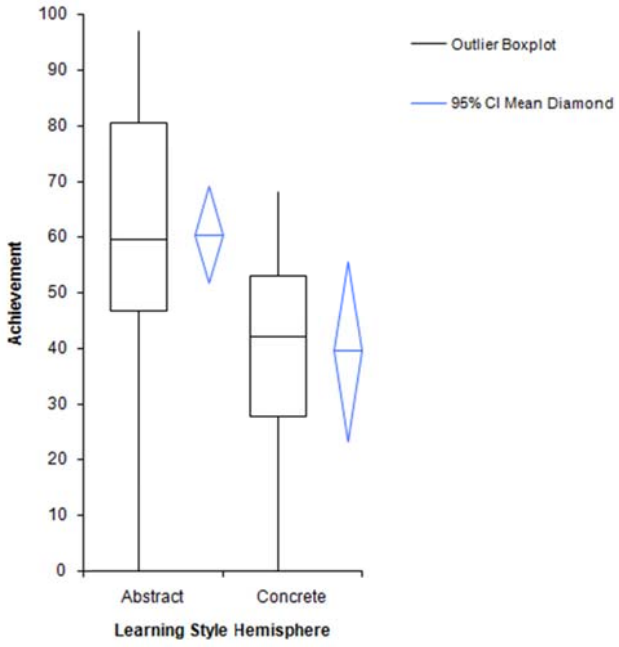


Figure 5-19. Achievement of students in abstract and concrete groups

This is one of the five occasions in this research when a significant hypothesis was found below a non-significant parent. It could be suggested that this is indicative of an error in the research, where the parent is a false negative and in this instance there are indicators to suggest that this may be the case. Firstly the ANOVA result was quite close to the boundary level ($F(3, 2.73) = 0.059$) and secondly the small number students in two of the

groups means that the power of the test was limited. However, it is quite conceivable that of the two measures that are used in constructing the KLS, the Perception continuum influences achievement, whilst the Processing continuum does not.

Figure 5-20. The LS-A hypothesis tree at the second level

5.5 Halting due to a fruitless search

The expansion of the tree halts when the search for interesting connections in the data appears to be fruitless. The search is defined as one in which there are no significant results found at both the current level and at the parent level in the hierarchical hypothesis dataset, or no reason can be found to continue.

As a consequence of the non-significant test result, "Does the number of events at week 3 affect achievement", the node at 1.2.1.1.1 in the B-A hypothesis tree is marked with an X and coloured green to indicate a non-significant result. The control algorithm directs that the search should progress for one more level so the child attributes are formed into hypothesis and added to the tree Figure 5-21. At a related node in the tree (1.1.1) five child attributes were formed into hypothesis but since the number of Module and Class events over the entire semester was small, these hypotheses were not included.

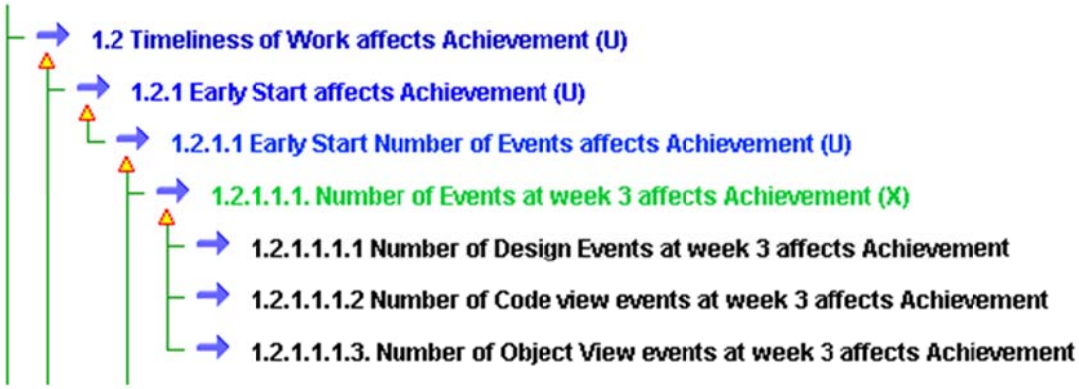


Figure 5-21. The B-A hypothesis tree at 1.2.1.1.1.1

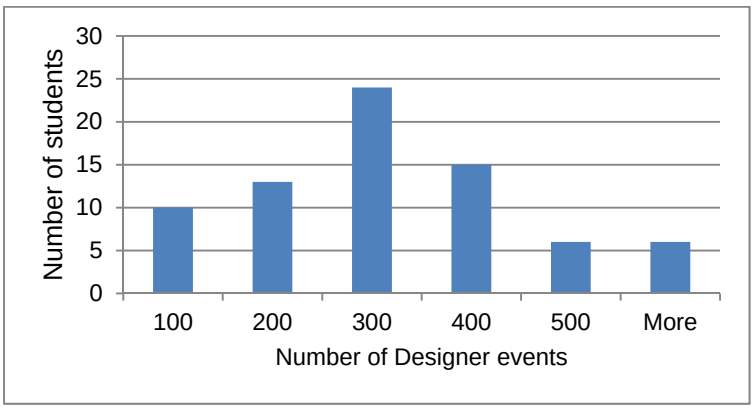


Figure 5-22. Frequency distribution of Designer events per student at week 3

A frequency distribution over the number of Designer events in Figure 5-22 shows an approximately normal distribution with most students using between 100 and 400 Designer events in this period.

A scatterplot of the number of designer events at week 3 in the semester against the achievement at the end of semester is shown in Figure 5-23.

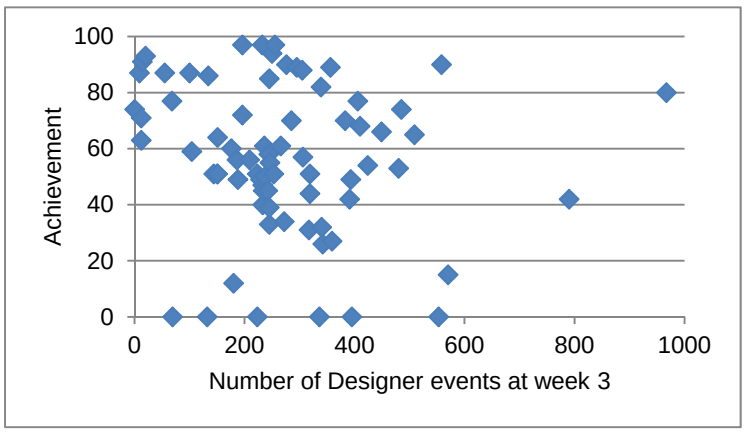


Figure 5-23. Designer events at week 3 against achievement

Tests are applied to the population to find out if there is evidence that they are not normally distributed. At the 0.05 level the lowest activity level group does have evidence of this but since the other four groups are satisfactory in this aspect, the ANOVA is applied. The correlation coefficient is -0.14, indicating no relationship between the attributes.

Table 5-8. Tests for 'non-normality' in Designer events at week 3 populations

Anderson-Darling A^2	1.12	0.64	0.65	0.35	0.61
p of A^2	0.004	0.076	0.072	0.428	0.089
Shapiro-Wilk W	0.80	0.92	0.89	0.94	0.89
p of W	0.003	0.167	0.071	0.343	0.087

A formal statement of the hypotheses is in Figure 5-24. This is then tested using ANOVA in the standard manner.

H_0: The number of designer events at week 3 does not affect achievement.

H_1: The number of designer events at week 3 affects achievement.

Figure 5-24. Statement of hypothesis 1.2.1.1.1.1

The result of the ANOVA test on the five groups (Table 5-9) gives an F value of 0.63. The probability of this result, assuming the null hypothesis, is almost 0.64. This means that the null hypothesis must be accepted and the conclusion reached that there is no evidence that the groups are not homogeneous. Figure 5-25 supports this finding and illustrates no clear differences between the groups.

Table 5-9. ANOVA result: Designer events at week 3 and achievement

Groups	**n**	**Mean**	**SE**	**Pooled SE**	**SD**
Low	15	65.1	7.71	6.93	29.9
Mid-Low	15	53.4	6.60	6.93	25.6
Mid	15	59.9	5.59	6.93	21.6
Mid-High	15	53.2	7.10	6.93	27.5
High	14	52.4	7.72	7.17	28.9
Source of variation	**Sum squares**	**DF**	**Mean square**	**F statistic**	**p**
Groups	1826.7	4	456.7	0.63	0.64
Residual	49650.7	69	719.6		
Total	51477.4	73			

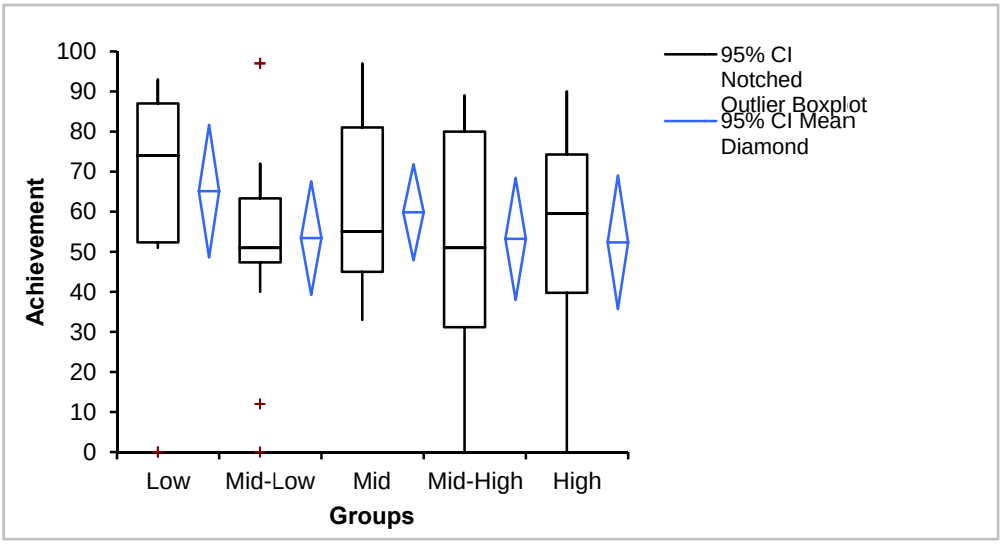


Figure 5-25. Mean and 95% CI for the mean for designer activity groups at week 3

As a result of the non-significant result of the hypothesis 1.2.1.1.1.1, this is encoded into the tree (Figure 5-26). The control algorithm directs that since this is the second consecutive non-significant result in this branch of the tree then the search should stop, although this is not otherwise a leaf of the tree, so the node of the tree is pruned at this point and notation (Table 5-1) added to the tree node.

Figure 5-26. The result of 1.2.1.1.1.1 is encoded into the tree

5.6 Halting due to insufficient data

At certain points in the application of the control algorithm to the data, there are insufficient data to continue the process. The B-A hypothesis tree in Figure 5-27 required a hypothesis test at node 1.1.1.2.3.3. that would test whether the number of show notes on/off events affected achievement.

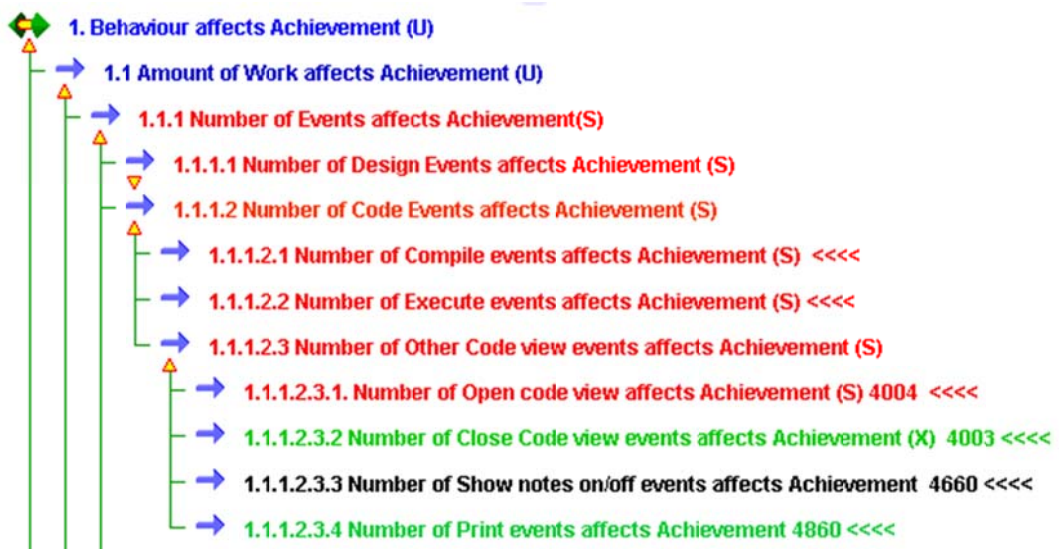


Figure 5-27. Hypothesis 1.1.1.2.3.3. in the B-A tree cannot be tested

Only 54 of the 74 students used this operation and approximately half of these used it five times or less during the semester. The method used in the rest of this hypothesis tree of dividing the students into five equal sized groups from low activity to high activity is not reasonable in this case because in the first two groups there would be several students with two operations in both of the lowest activity groups. Therefore no further analysis was carried out.

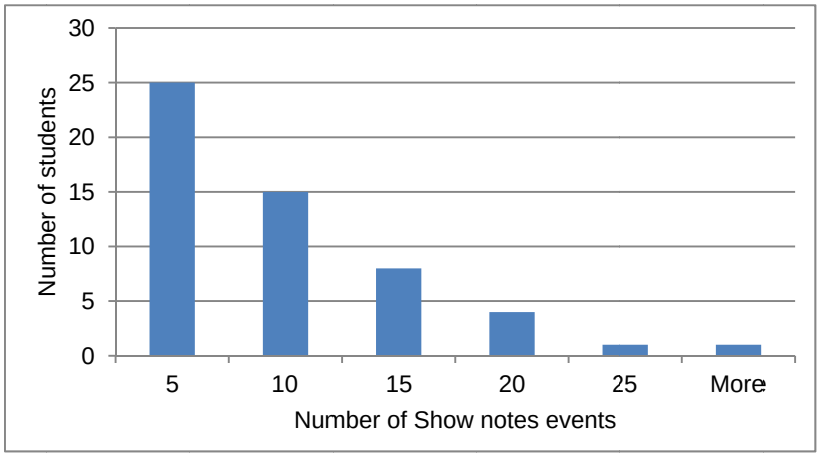


Figure 5-28. Frequency distribution of students

5.7 The complete hypothesis trees

The three hypothesis trees that have been created as a result of the application of the control algorithm are presented in this section. Each tree

was created by the application of the control algorithm to a primary hypothesis.

The three hypothesis trees are:

1. The Learning style – Achievement (LS-A) tree which investigated the relationship between the 2-D attribute Learning style and the 1-D attribute Achievement (see 5.7.1)
2. The Learning style – Behaviour (LS-B) tree which investigated relationships between two attributes that have more than one dimension, Learning style is 2-D and Behaviour is *n*-D (see 5.7.2).
3. The Behaviour – Achievement (B-A) is the result of investigating the relationship between the *n*-D attribute Behaviour and the 1-D attribute Achievement; it is an extensive tree that is also quite deep because of the significant results that were found (see 5.7.3).

Following the application of the control algorithm, when each hypothesis tree was complete, it was considered useful to find a measure of the relative importance of the sub-trees. This measure would allow for a comparison between sub-trees and was required to carry out the validation process that is defined in section 6.1.

This proved to be a difficult task and one that has been partially solved by using two measures, the significance ratio and density percentage. These were added to each node of the complete trees in an attempt to provide a means of comparing the relative importance of sibling nodes using the significant hypothesis as a basis.

The significance ratio is the number of significant hypothesis in the subtree, to the total number of hypothesis in the subtree. Thus this ratio and the calculated percentage are measures of the density of significant hypotheses in any subtree. In the process of formulating an appropriate measure, there were some questions that arose on precisely how this should be done. Two important questions were:

1. Should the hypotheses that were not tested because insufficient data was available (the black nodes) be included in the count?

The black nodes were excluded, so only nodes where the hypothesis test was carried out were included in the count. As a result at node 1.2.2.1.3 which is the root of the subtree in Figure 5-29, only the green nodes are counted and the ratio is 0/3.

2. At any level, were the child hypotheses and the node itself given equal weight?

Referring again to Figure 5-29, a node and its child nodes are simply counted in the percentage and ratios. The objective being to compare relative weights of sibling subtrees and so other more complex approaches were not considered necessary.

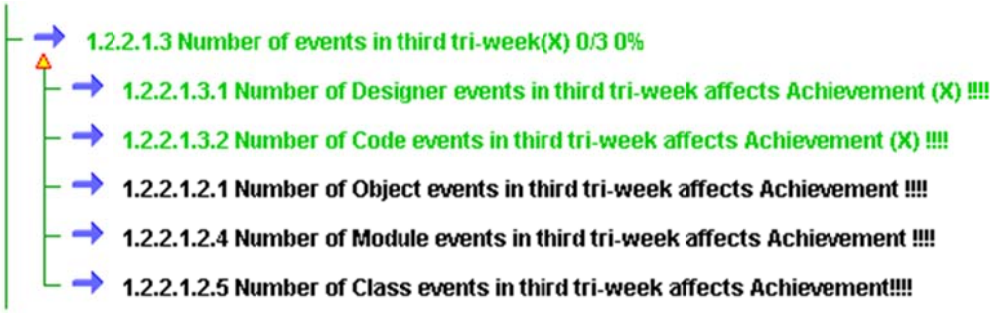


Figure 5-29. A subtree to illustrate density percentage and significance ratio

These metrics are an attempt to define the relative importance of subtrees. However, there is a potential issue that is demonstrated here with the use of theoretical trees. The tree in Figure 5-30 has a root node where the significance ratio is 1/3 and the density percentage is 33%. The tree has two subtrees which are pruned and are therefore remain leaves of the tree. They have extreme values for the ratios and density.

Figure 5-30. Theoretical tree A

A similar tree is illustrated in Figure 5-31. The main difference is that in this tree, rather than being pruned at 1.1, three child hypotheses were added and found to be not significant. The result is that the significance ratio of node 1.1. is greatly reduced and that of node 1 appreciably reduced. So the significance ratio and density percentage should be used with care.

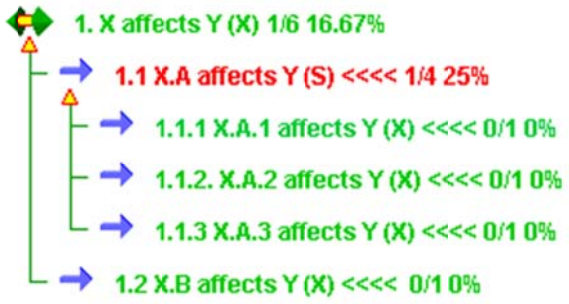


Figure 5-31. Theoretical tree B

Another interpretation of the density is as a measure of information content. Each hypothesis that is found to be significant contributes to the known information. The absolute values are therefore less useful but it is envisioned that the comparison of similar subtrees will be useful.

The purpose of the density percentage and significance ratio is to provide a measure of the relative importance of the subtree to the parent hypothesis. A problem is the ambiguity when sub trees of considerably different sizes have the same density. Hence, the retention of the raw numeric values in the weight/size ratio, to provide additional information. This is helpful because for nodes with very small numbers of children the measures are highly variable.

The two measures selected therefore have some weaknesses but, despite their limitations, no better measures were found, and so these measures will be applied and discussed whilst noting that the conclusions should be used with care.

Even though not perfect, this information may be useful because it can demonstrate which branches of the tree hold the significant hypotheses even when the trees are displayed rolled-up so some of the details are not visible.

Figure 5-32. Hypothesis tree with the significance ratio and density percentage

A straight forward example of the weight and density calculation can be seen in Figure 5-32. This tree shows that 46% of the *Amount of work*

hypotheses that were tested were significant with respect to Achievement, 50% of the *Timeliness of work* hypotheses were significant and 57% of the *Effectiveness of work* hypothesis were significant. This could be an indication that the *Effectiveness of work* (1.3) is in some sense more important with respect to achievement than the *Timeliness of work* (1.2) and both are more important than the *Amount of work* (1.3). This will be discussed further in Chapter 6 when groups of significant hypotheses are investigated.

5.7.1 Learning style - achievement (LS-A)

The LS-A tree (Figure 5-33) is an example of a hypothesis tree that could not grow even if all hypothesis testing produced significant results because few child attributes are available for Learning style, and Achievement has only one dimension. The LS-A hypothesis tree is by far the smallest of the trees and consists of only three nodes, one of which has a significant result. Since, this was found below a non-significant result it is an indicator of the worth of continuing the process to one more level when a non-significant result is found.

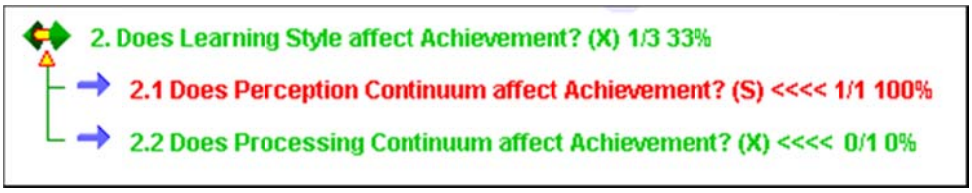


Figure 5-33. The complete LS-A hypothesis tree

5.7.2 Learning style - behaviour (LS-B)

The LS-B hypothesis tree had two *n*-D attributes under investigation and hence the addition of child attributes, under the direction of the control algorithm, caused the number of nodes in the tree to grow quickly. This tree however did not present any significant results in the hypothesis testing. The requirement of the algorithm to continue to two levels in this tree seemed tedious, the more so as additional non-significant results were found. The complete LS-B hypothesis tree is in Figure 5.34.

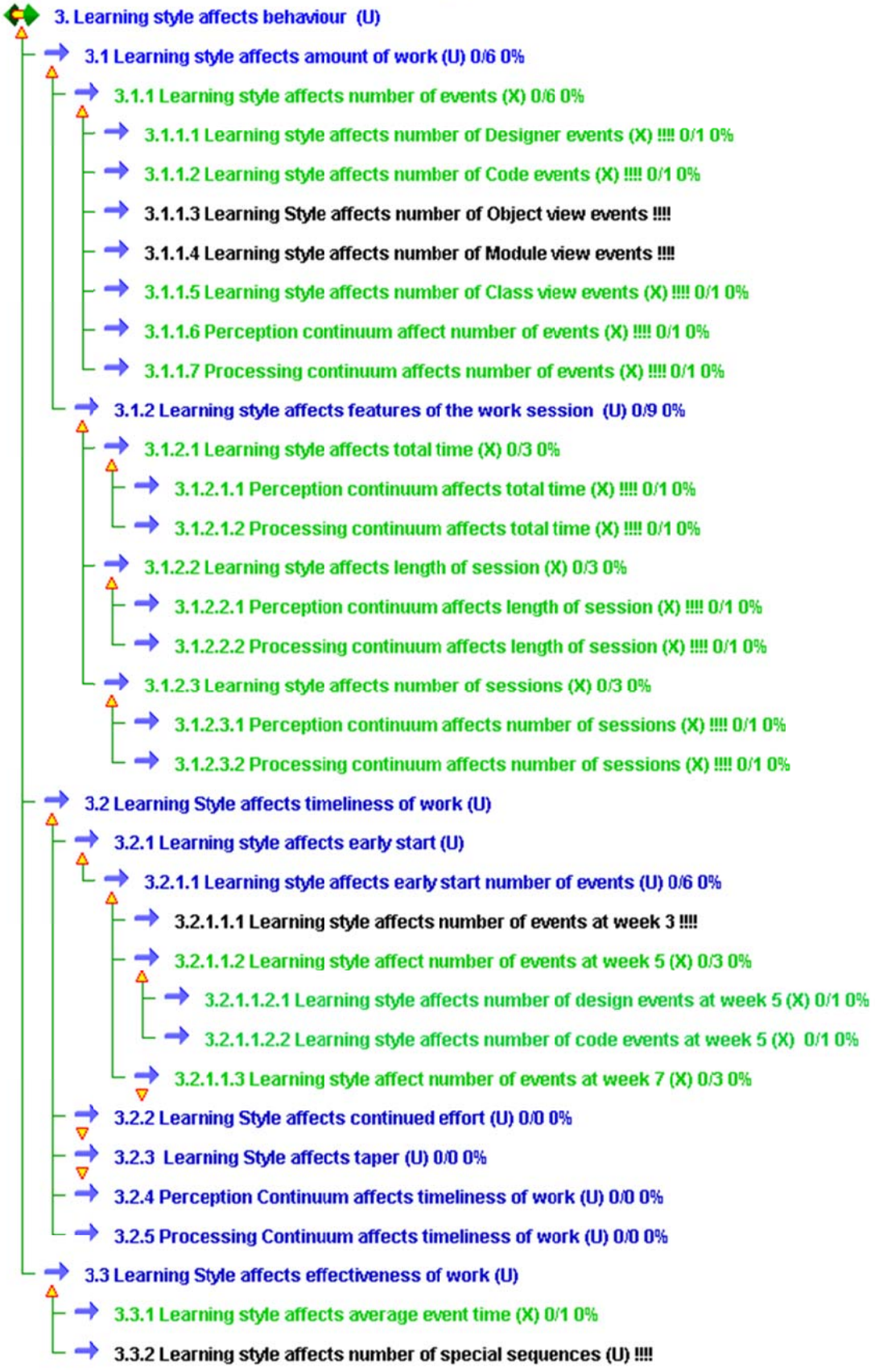
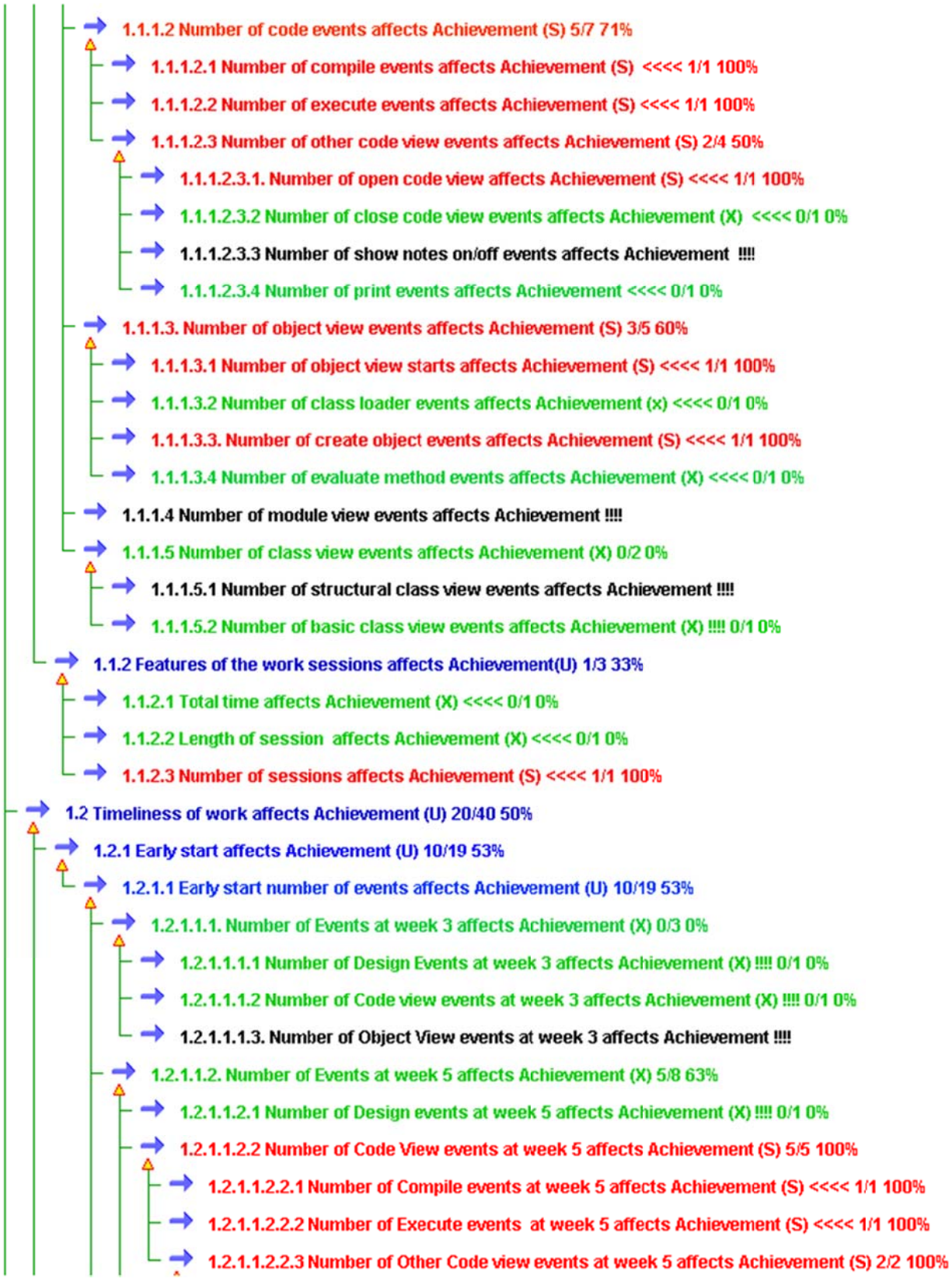


Figure 5-34 The complete LS-B hypothesis tree

5.7.3 Behaviour - achievement tree (B-A)

Since there were many facets of Behaviour that could be tested and some significant results were found, this hypothesis tree became quite extensive. There were ten nodes where the control algorithm provided for the tree to be pruned which suggested that pruning was worthwhile. There were only four (2.6%) significant nodes that were discovered below a non-significant result that suggests that it was probably not really useful to continue to one more level as directed by the control algorithm.

The complete hypothesis tree is in Figure 5-35.


1.1.1.2 Number of code events affects Achievement (S) 5/7 71%
1.1.1.2.1 Number of compile events affects Achievement (S) <<<< 1/1 100%
1.1.1.2.2 Number of execute events affects Achievement (S) <<<< 1/1 100%
1.1.1.2.3 Number of other code view events affects Achievement (S) 2/4 50%
1.1.1.2.3.1. Number of open code view affects Achievement (S) <<<< 1/1 100%
1.1.1.2.3.2 Number of close code view events affects Achievement (X) <<<< 0/1 0%
1.1.1.2.3.3 Number of show notes on/off events affects Achievement !!!!
1.1.1.2.3.4 Number of print events affects Achievement <<<< 0/1 0%
1.1.1.3. Number of object view events affects Achievement (S) 3/5 60%
1.1.1.3.1 Number of object view starts affects Achievement (S) <<<< 1/1 100%
1.1.1.3.2 Number of class loader events affects Achievement (x) <<<< 0/1 0%
1.1.1.3.3. Number of create object events affects Achievement (S) <<<< 1/1 100%
1.1.1.3.4 Number of evaluate method events affects Achievement (X) <<<< 0/1 0%
1.1.1.4 Number of module view events affects Achievement !!!!
1.1.1.5 Number of class view events affects Achievement (X) 0/2 0%
1.1.1.5.1 Number of structural class view events affects Achievement !!!!
1.1.1.5.2 Number of basic class view events affects Achievement (X) !!!! 0/1 0%
1.1.2 Features of the work sessions affects Achievement(U) 1/3 33%
1.1.2.1 Total time affects Achievement (X) <<<< 0/1 0%
1.1.2.2 Length of session affects Achievement (X) <<<< 0/1 0%
1.1.2.3 Number of sessions affects Achievement (S) <<<< 1/1 100%
1.2 Timeliness of work affects Achievement (U) 20/40 50%
1.2.1 Early start affects Achievement (U) 10/19 53%
1.2.1.1 Early start number of events affects Achievement (U) 10/19 53%
1.2.1.1.1. Number of Events at week 3 affects Achievement (X) 0/3 0%
1.2.1.1.1.1 Number of Design Events at week 3 affects Achievement (X) !!!! 0/1 0%
1.2.1.1.1.2 Number of Code view events at week 3 affects Achievement (X) !!!! 0/1 0%
1.2.1.1.1.3. Number of Object View events at week 3 affects Achievement !!!!
1.2.1.1.2. Number of Events at week 5 affects Achievement (X) 5/8 63%
1.2.1.1.2.1 Number of Design events at week 5 affects Achievement (X) !!!! 0/1 0%
1.2.1.1.2.2 Number of Code View events at week 5 affects Achievement (S) 5/5 100%
1.2.1.1.2.2.1 Number of Compile events at week 5 affects Achievement (S) <<<< 1/1 100%
1.2.1.1.2.2.2 Number of Execute events at week 5 affects Achievement (S) <<<< 1/1 100%
1.2.1.1.2.2.3 Number of Other Code view events at week 5 affects Achievement (S) 2/2 100%


1.2.1.1.2.2.3.1 Number of Open code view at week 5 affects Achievement (S) <<<< 1/1 100%
1.2.1.1.2.2.3.2 Number of Close Code view events at week 5 affects Achievement !!!!
1.2.1.1.2.2.3.3 Number of Print events at week 5 affects Achievement !!!!
1.2.1.1.2.3 Number of Object View Events at week 5 affects Achievement (X) !!!! 0/1 0%
1.2.1.1.3. Number of Events at week 7 affects Achievement (X) 5/8 63%
1.2.1.1.3.1 Number of Design events at week 7 affects Achievement (X) !!!! 0/1 0%
1.2.1.1.3.2 Number of Code View Events at week 7 affects Achievement (S) 5/5 100%
1.2.1.1.3.2.1 Number of Compile Events at week 7 affects Achievement (S) <<<< 1/1 100%
1.2.1.1.3.2.2 Number of Execute Events at week 7 affects Achievement (S) <<<< 1/1 100%
1.2.1.1.3.2.3 Number of Other Code Events at week 7 affects Achievement (S) 2/2 100%
1.2.1.1.3.2.3.1 Number of Open code view at week 7 affects Achievement (S) <<<< 1/1 100%
1.2.1.1.3.2.3.2 Number of Close Code view events at week 7 affects Achievement !!!!
1.2.1.1.3.2.3.3 Number of Print events at week 7 affects Achievement !!!!
1.2.1.1.3.3 Number of Object View Events at week 7 affects Achievement (X) 0/1 0%
1.2.2 Continued Effort affects Achievement (U) 10/20 50%
1.2.2.1 Number of events in three week blocks affects Achievement (U) 10/18 56%
1.2.2.1.1.Number of events in first tri-week (X) !!!! 0/1 0%
1.2.2.1.2.Number of events in second tri-week (S) 5/6 83%
1.2.2.1.2.1 Number of Designer events in second tri-week affects Achievement (X) !!!! 0/1 0%
1.2.2.1.2.2 Number of Code events in second tri-week affects Achievement (S) 4/4 100%
1.2.2.1.2.2.1 Number of Compile events in second tri-week affects Achievement (S) <<<< 1/1 100%
1.2.2.1.2.2.2 Number of Execute events in second tri-week affects Achievement (S) <<<< 1/1 100%
1.2.2.1.2.2.3 Number of Other code events in second tri-week affects Achievement 1/1 100%
1.2.2.1.2.2.3.1 Number of Open code view events in second tri-week affects Achievement !!!!
1.2.2.1.2.2.3.2 Number of Close Code view events in second tri-week affects Achievement !!!!
1.2.2.1.2.2.3.3 Number of Show notes on/off events in second tri-week affects Achievement !!!!
1.2.2.1.2.2.3.4 Number of Print events in second tri-week affects Achievement !!!!
1.2.2.1.2.3 Number of Object events in second tri-week affects Achievement !!!!
1.2.2.1.2.4 Number of Module events in second tri-week affects Achievement !!!!
1.2.2.1.2.5 Number of Class events in second tri-week affects Achievement !!!!
1.2.2.1.3 Number of events in third tri-week(X) 0/3 0%
1.2.2.1.3.1 Number of Designer events in third tri-week affects Achievement (X) !!!! 0/1 0%
1.2.2.1.3.2 Number of Code events in third tri-week affects Achievement (X) !!!! 0/1 0%
1.2.2.1.2.1 Number of Object events in third tri-week affects Achievement !!!!

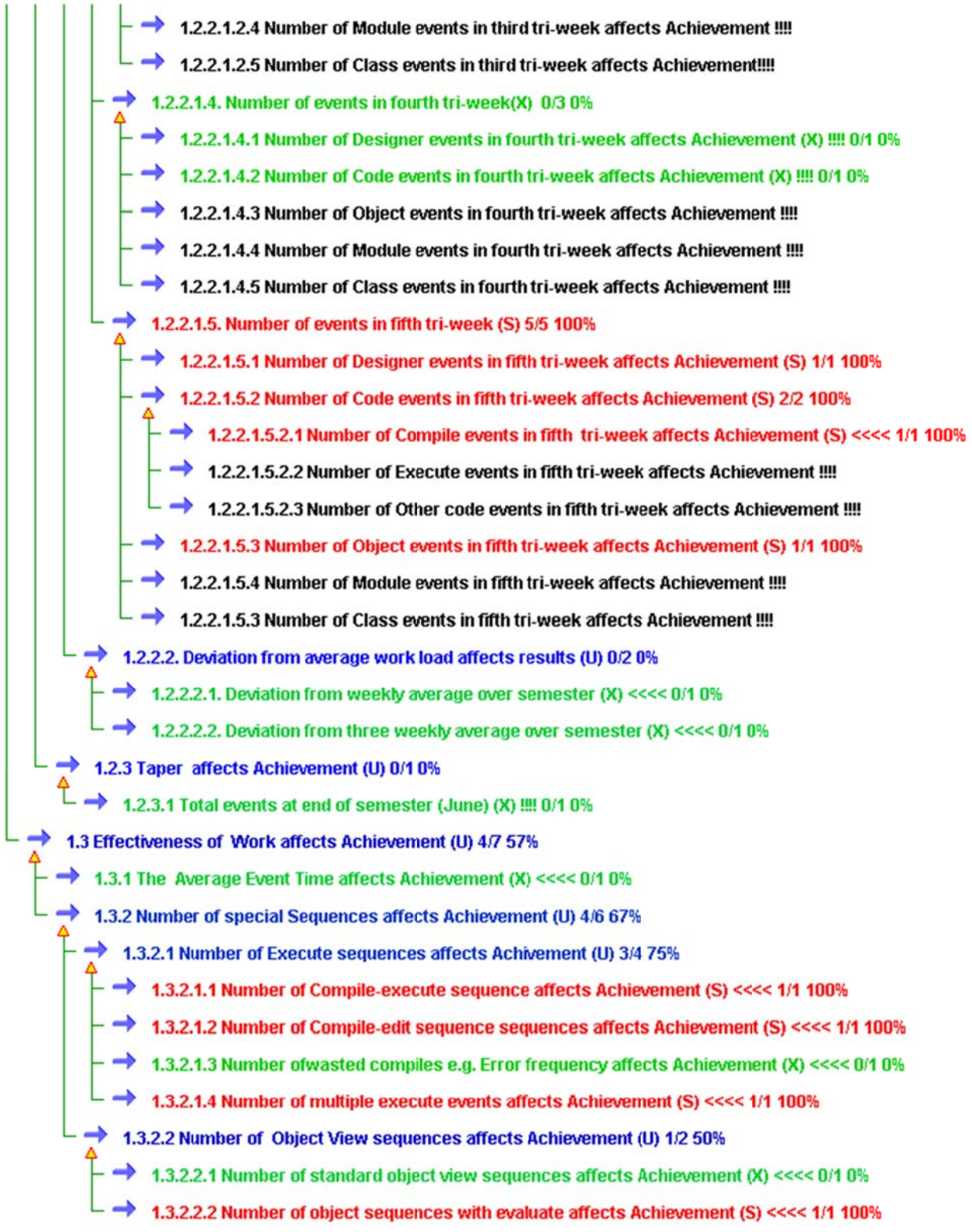


Figure 5-35. The complete B-A hypothesis tree

5.7.4 A summary of the hypothesis trees

The application of the control algorithm to the three primary hypotheses has resulted in the three hypothesis trees. Table 5-10 is a summary of each of the complete hypothesis trees.

Table 5-10. Summary of the complete hypothesis trees

Statistic	LS-A		LS-B		B-A	
Maximum depth	2		4		8	
Number of nodes	3	100%	24	100%	123	100%
Number of significant nodes (S)	1	33.3%	0	0%	41	33.3%
Number of non-significant nodes (X)	2	66.7%	10	41.6%	43	35%
Number of unquantifiable nodes (U)	0	0%	7	29.2%	14	11.4%
Number of nodes not tested due to insufficient data	0	0%	0	0%	25	20.3%
Number of prunings (!!!!)	0	0%	3	12.5%	13	8.1%
Number of leaf nodes (<<<<)	2	66.7%	3	12.5%	45	29.8%
Number of significant nodes with non-significant parent over number of significant nodes	1	100%	0	0%	4	2.6%
Number of significant nodes with significant parent	0	0%	0	0%	29	19.2%

5.8 Operational issues in the application of the control algorithm

The two purposes of the control algorithm were to:

- ensure that all relevant aspects of the available data were presented for investigation through hypothesis generation and testing
- limit the potential for combinatorial explosion

The application of the algorithm was not completely automated since there are several situations where this process requires domain knowledge. The following decision points in the application of the algorithm have been identified as requiring the application of this knowledge.

1. Defining a node as unquantifiable
2. Preventing repetition in related hypotheses
3. Specifying and carrying out the hypothesis test
4. Selecting the child attributes
5. Stopping the search through pruning

These are all situations where intervention is required on the part of the researcher and each one is now explained.

5.8.1 Defining a node as unquantifiable

A node was classed as unquantifiable if there was no available measure for one of the attributes in the pertinent hypothesis. This apparent impasse was broken simply by the normal application of the CAP which directed the disaggregation of both attributes into child attributes (section 5.8.4), followed by the forming of new hypothesis which were added to the tree.

5.8.2 Preventing repetition in related hypotheses

It was noted in section 4.5.4 that related hypothesis will occur throughout the hypothesis tree because the method of expansion led to subsets of data being used. Domain knowledge must be applied to recognise how closely the hypotheses are related. For example in the B-A tree (Figure 5-35) node *1.2.1.1 Early start number of events affects Achievement* has several child nodes below it in the tree. This series of tests was initially planned to find out whether the events were significant at various stages during the semester. It was planned to look at the cumulative number of events at weeks 3 and 5. However, all the week three results were either not significant or there was insufficient data to carry out a test. At week 5, the Code view events were found to be significant. These events were already known to be significant at the end of semester, so the testing was also carried out for week 7 to see if the same pattern applied.

In the LS-B tree (Figure 5-34), it seems futile to test the series of three child hypothesis of 3.2.1.1 (Figure 5-36) since none of the previous hypotheses in the tree has been found to be significant. Knowledge of the data would indicate that it is appropriate to prune the tree at 3.2.1 and so this branch of the tree is fruitless.

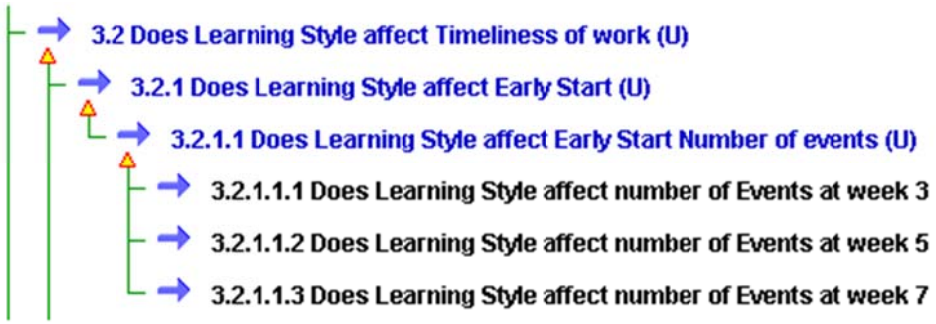


Figure 5-36. New nodes are related to previous tests in the LS-B tree

5.8.3 The hypothesis test and statistics

The precise nature of the hypothesis and the statistical test used may have affected the outcome of some of the hypothesis testing. As a consequence of this, the resultant hypothesis tree could have been different.

The statistical test chosen in most of this research was ANOVA. This was because of the decision to consider differences between more than two groups. In some cases the result would have been very likely to be different if the research had looked for differences between two groups e.g. only the highest activity levels and lowest. In fact this statistic was used for testing between the hemispheres of KLSI in the LS-A tree.

It is also possible that different results would have been achieved with different groups in the ANOVA, for example at node *1.1.2.1 Total time affects achievement*, the hypothesis tested related total number of hours programming with achievement and was tested with ANOVA over 5 groups. The resulting F value was 2.27 and this had a probability of 0.07, since this was just above the selected critical value of 0.05 then the null hypothesis must be accepted and a non-significant result recorded (Table 5-11).

Figure 5-37 indicates that, despite the non-significant result, there does appear to be some tendency for those with higher number of hours programming to achieve higher exam results. This same hypothesis has been reworked with three groups instead of five to illustrate how different results could have arisen if alternate decisions had been made, although it should be noted that it is clearly inappropriate to search for a method that will produce a significant result.

Table 5-11. Total time affects achievement in five groups

Groups	n	Mean	SE	Pooled SE	SD
Low	15	40.9	7.79	6.63	30.2
Low-mid	15	53.6	7.43	6.63	28.8
Mid	15	63.6	5.77	6.63	22.4
High-mid	15	61.5	5.28	6.63	20.4
High	14	65.2	6.76	6.86	25.3
Source of variation	**Sum squares**	**DF**	**Mean square**	**F statistic**	**p**
Groups	5982.3	4	1495.6	2.27	0.07
Residual	45495.0	69	659.3		
Total	51477.4	73			

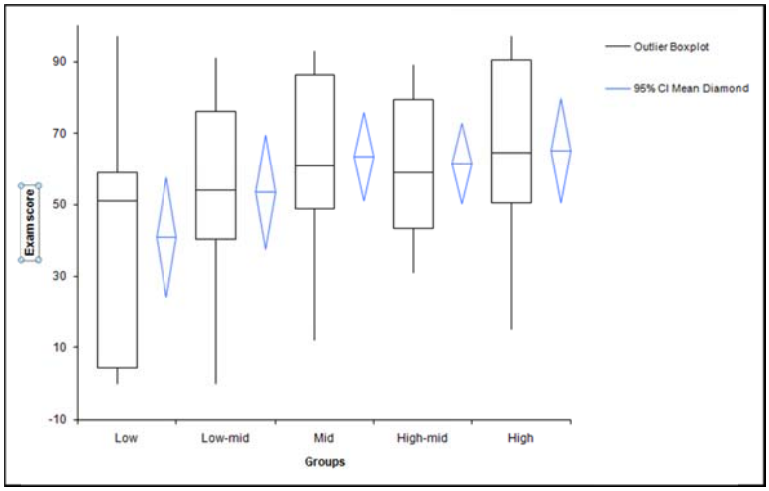


Figure 5-37. Mean and 95% CI for the mean at *1.1.2.1 Total time affects achievement*

If only three groups were used, the number of students in each activity level group increases to twenty-five. The ANOVA results has an F value of 3.113 and the probability of a difference is 0.05, exactly equal to the critical value of 0.05 (Table 5-12).

What has been shown is that a borderline situation has been reached with three groups and a non-significant result with five groups. This clearly does not demonstrate that the number of groups should be chosen to obtain significant results but merely that a different result could be obtained given a slightly different application of the test. Which version should be used depends on the objective of the research. If the purpose had been to look for

differences in achievement between the highest 50% and the lowest 50% then it would have been appropriate to use two groups, but the research was using five groups from Low through Mid to High activity levels and sought to find if there was a significant difference between these five groups. If a significant difference is found between some activity groups, it is then worthwhile to find out between which groups these differences occur.

Table 5-12. Total time affects achievement in three groups

Groups	n	Mean	SE	Pooled SE	SD
Low25	25	46.6	6.33	5.16	31.6
Mid25	25	60.1	4.37	5.16	21.8
High25	24	64.2	4.64	5.27	22.7

Source of variation	Sum squares	DF	Mean square	F statistic	p
Groups	4172.2	2	2086.1	3.13	0.05
Residual	47305.2	71	666.3		
Total	51477.4	73			

Figure 5-38 shows a rising mean with each activity level group.

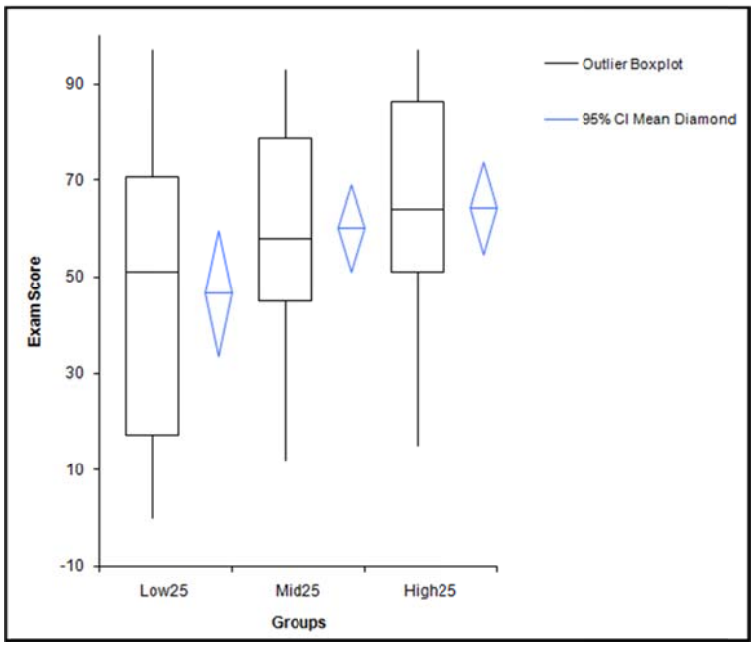


Figure 5-38. Total time affects achievement in three groups

When the five groups have been found to be significantly different, the relationships between the dependent and independent variables has not

always been the same. Post-hoc testing has revealed the groups that were found to be significantly different and this has not always been between the highest and the lowest activity groups. Situations have been uncovered where just the lowest activity group achieved poor results (a deficit situation), where achievement was monotonically increasing with activity levels (the more the better) and where the highest achievement was in the middle activity group (too much of a good thing).

Thus using five groups was useful in determining those hypotheses that were not simply higher activity led to higher achievement but where the highest achievement was at intermediate levels. This would have probably not have been visible if only three groups had been used. When the highest activity levels did not have the highest mean achievement, the hypothesis is of particular interested in terms of teaching and learning, because an activity has been identified where the highest users are not the highest achievers.

The selection of the statistical tests to be used in this research was discussed in section 4.9. The significance level of a statistical test is the probability of rejecting the null hypothesis when it is true and in this research has been set to 0.05. There is a trade-off between type I and type II errors given a certain sample size. The power of a statistical test is its ability to reject a null hypothesis when it is false and the maximum value of 1 is the ideal and unobtainable value, since the definition of power is 1 minus the probability of a type II error. In this research a type II error would indicate that a null hypothesis has been accepted and the conclusion reached that there is no effect when the null hypothesis should have been rejected. The effect of this would be to overlook relationships in the data.

However in this research the sample size was fixed, the entire population of students was used and it was not possible to increase it. Thus the researcher had no way of influencing the probability of a type II error, once the probability of a type I error was selected. In this research a relatively high alpha value was chosen (0.05) because in this exploratory research where potential relationships are being sought, Type I errors were on balance considered slightly more acceptable than Type II errors.

5.8.4 Selecting the child attributes

If both the attributes in a hypothesis were 1-D then the node is a leaf of the tree and marked (<<<<) thus indicating that no child attributes were available. Otherwise a crucial part of the control algorithm is the identification and selection of child attributes which are used to form additional hypotheses. These are then placed into the hypothesis tree as children of the existing node.

The child attributes that were selected are not necessarily the only possible ones that could have been used. For example, when seeking the child attributes of the total number of events in the B-A tree at level 1.1.1, the five children chosen are related to the use of different views in the programming environment. The views appeared a natural division; however these could have been divided in other groupings including perhaps joining some of the sub groups and several other versions. In general the child attributes were selected by the application of domain knowledge and in some cases the judgement of the researcher.

5.8.5 Stopping the search through pruning

One important issue in the design of the control algorithm was at what point to stop the expansion of the tree. In favour of stopping as soon as possible, was the need to limit the overall size of the tree and in favour of continuing to expand the tree as much as possible, was the desire to uncover all interesting relationships in the data. The stopping case used was after finding two levels - a parent and child - where the result was not significant.

The question raised here, is whether the pruning was done too soon or not soon enough. It would be possible to prune earlier, that is, to cease the expansion as soon as a non-significant result was found. Had this been done, then some potentially interesting results would not have been found, in the B-A and LS-A hypothesis trees but some repetitive testing would not have been required in the LS-B tree.

In fact only five situations were found where significant hypothesis were uncovered below significant ones; four of these in the B-A tree and one in the LS-A tree (discussed in section 5.7.1).

One interpretation of the non-significant hypotheses 1.1.1.1.4 and 1.1.1.1.5 in Figure 5-39 and 1.2.1.1.2 and 1.2.1.1.3 in Figure 5-40, is that they are false negatives (or Type II errors) because they can be seen as being a part of a path of significant hypotheses down the tree. Careful consideration of the four such hypotheses in the B-A tree however does not ratify this conclusion.

Hypothesis *1.1.1.1.4 Number of Edit Node events affects Achievement* was not found to be significant. This hypothesis represents a group of events in P-Coder's Designer view where students are adding, editing or viewing the detail behind a line of pseudocode. The child nodes that are below this level in the tree are three hypotheses that test three edit events separately and only the event that actually updates the information is found to be significant (1.1.1.1.4.2), while simply opening or viewing the detail was not.

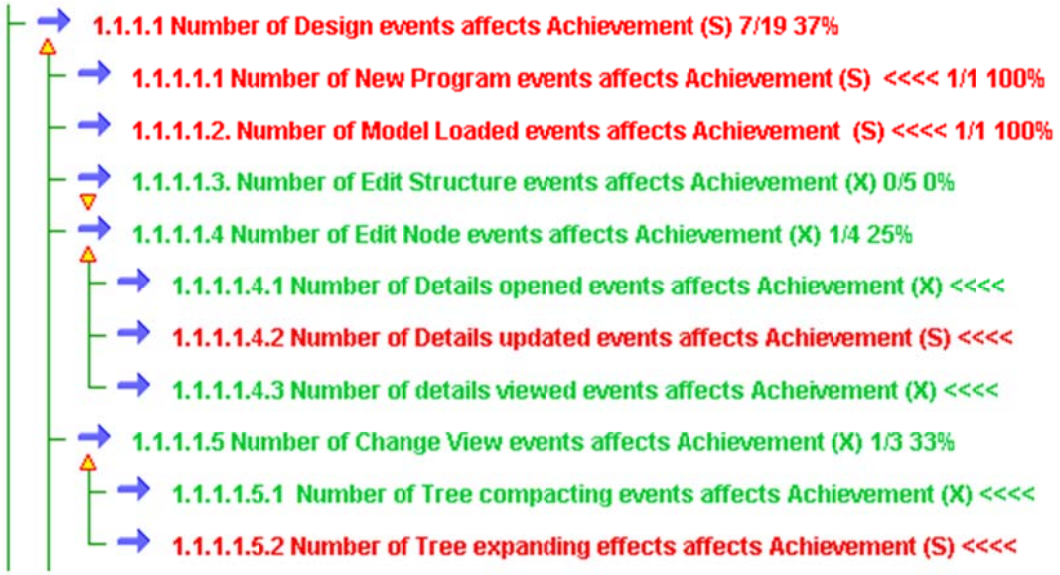


Figure 5-39. Design events subtree: two significant hypotheses at second level

There is a similar situation in Figure 5-40, where two significant hypotheses are below non-significant parent nodes. In this case the non-significant hypothesis tested for a relationship between the total number of events at a particular week of the semester (week 5 for 1.2.1.1.2 and week 7 for 1.2.1.1.3)

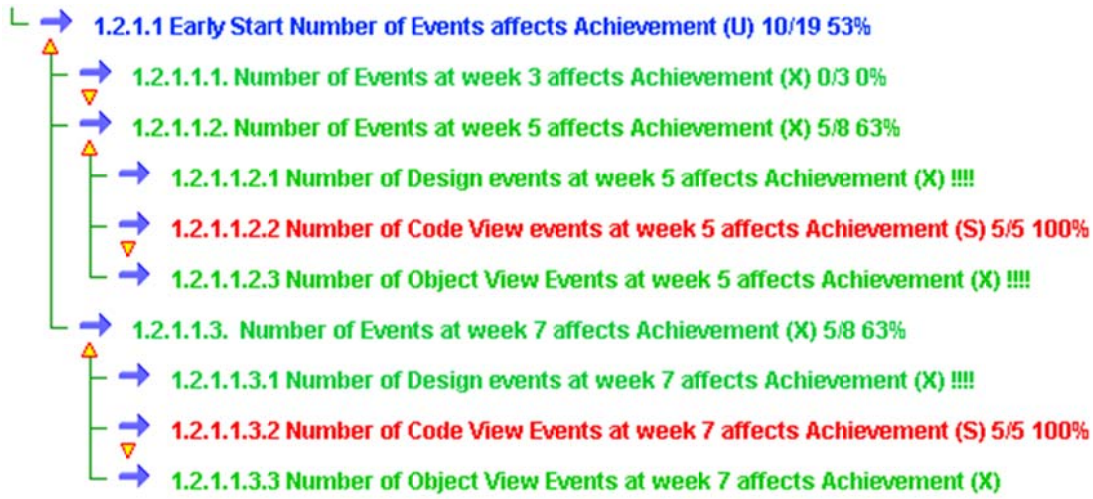


Figure 5-40. Early start subtree: two more significant hypotheses at second level

5.9 Summary

This chapter has demonstrated examples of the application of the control algorithm to the data through a series of cases that displayed the key decision points in the algorithm.

Three hypothesis trees have been created, each one related to a research question. Each of the hypothesis trees has been presented and a numerical summary of these shows the differences in the trees. This occurred because of differences in the nature of the data and the results of hypothesis testing that were achieved.

The later part of the chapter critiqued operational issues in the application of the control algorithm. Chapter 6 will discuss the outcomes of the application of the research model presented and provide a means of dealing with the complexity of this number of results.

CHAPTER 6 ANALYSIS AND DISCUSSION

This chapter will analyse the results from the application of the research model and discuss implications for the model in the light of these. The rationale is to evaluate the model.

Three schemes for classifying the significant hypotheses are described and will be followed by an assessment of the value of each of them. Different schemes are used to identify whether any one of them provides a different emphasis for the findings. Section 6.2 discusses the value of the classification schemes and the consequences of these for the model.

6.1 Schemes for classifying significant hypotheses

The three hypothesis trees that were created contain 42 significant hypotheses. This section explores three schemes for classifying them. The three schemes are:

1. The tree structure

   The tree structure was derived by application of the control algorithm to the primary hypotheses. The branches of the trees contain hypotheses that are inherently related because of the way that the tree was constructed; top-down, expanding the tree by disaggregation of attributes into child attributes (see section 4.6).

   Depending on the level of granularity required the significant hypotheses could be grouped or classified at any level in the tree. This is explained in section 6.1.1.

2. The tool structure

   The programming tool that provided the behaviour data that was used in this research, P-Coder, was discussed in detail in Chapter 3. Recall from section 3.4 that the tool has five views in which the programmer may approach different aspects of the software

development process. The five views are: Designer, Code, Class, Object and Module. In this scheme, the hypotheses are classified according to the relevant view of the tool.

A few hypotheses apply to more than one event and could therefore apply to two or more views, and in some cases a hypothesis could cover all of the views e.g. Does the total number of events affect achievement? Those hypotheses that apply to two views have been placed in both categories, while hypotheses that relate to events across all views have been placed into an overall category.

3. The literature

This scheme classifies the significant hypotheses according to aspects of programming that have been identified in the CSE literature as important in the learning process. The relevant set of the literature associates some aspect of programming behaviour to achievement. A great deal of this literature considers specific features of a student that he or she brings to the learning process, such as prior scores in Mathematics or Science; whereas this research is interested in the behaviour of students during a semester of their studies and whilst in the programming process. In selecting literature, it was not considered essential that it used empirical analysis; position papers were deemed appropriate. There were some significant hypotheses that could not be classified using this scheme and others that belonged in more than one group.

It should be noted that (to the researcher) these are the three most obvious schemes for classifying the hypotheses but there may well be others. Each of the three will be examined, along with some preliminary conclusions that may be drawn arising from the use of these classification schemes.

Three different ways of viewing the results are important to this research because if a degree of consistency is found between them then this would assist in validating the model. In this research both the tree structure and the tool (P-Coder) are specific to this research whilst the literature is, in a sense, a control group since the literature provides the “expert” knowledge.

6.1.1 Use the tree structure

Of the three hypothesis trees, only the Behaviour-Achievement (B-A) tree had a relatively large number of significant results (41); the Learning style-Behaviour (LS-B) tree had none and the Learning style-Achievement (LS-A) tree only one. So, the necessity to classify significant hypotheses is restricted to the B-A tree and only this one will be included in this discussion. The significant hypothesis from the LS-A tree will be discussed in 6.1.1.6.

The tree structure provides inbuilt groups of hypotheses in its sub-trees, since each one consists of hypotheses that are associated because of the way the tree was created. If this scheme is selected, the classifications can be formed at any level in the tree. The result of classifying to the second (Figure 6-1) and third levels (Figure 6-2) is shown to demonstrate two possibilities.

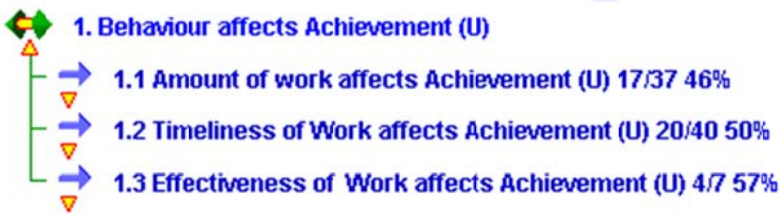


Figure 6-1. The B-A hypothesis tree at level two

Classifying at tree level two, divides the 41 significant hypotheses into three categories; two of these are of approximately equal size (1.1 and 1.2) and the third (1.3) is smaller. Recall from section 5.7 that the significance ratio and density percentages, at a node in the tree, show the ratio of significant hypotheses to total hypotheses and the equivalent percentage of the particular sub-tree. The percentage of significant hypotheses on each branch of the tree is approximately equal at this level of grouping (Figure 6-1). However, expanding the tree to one more level exposes more information without becoming too unwieldy and is therefore preferred (Figure 6-2).

In the sections that follow, each of the five level three categories that have significant hypotheses will be discussed. The level of the node of the tree (e.g. 1.1.1) that is to be discussed is indicated in the section heading text.

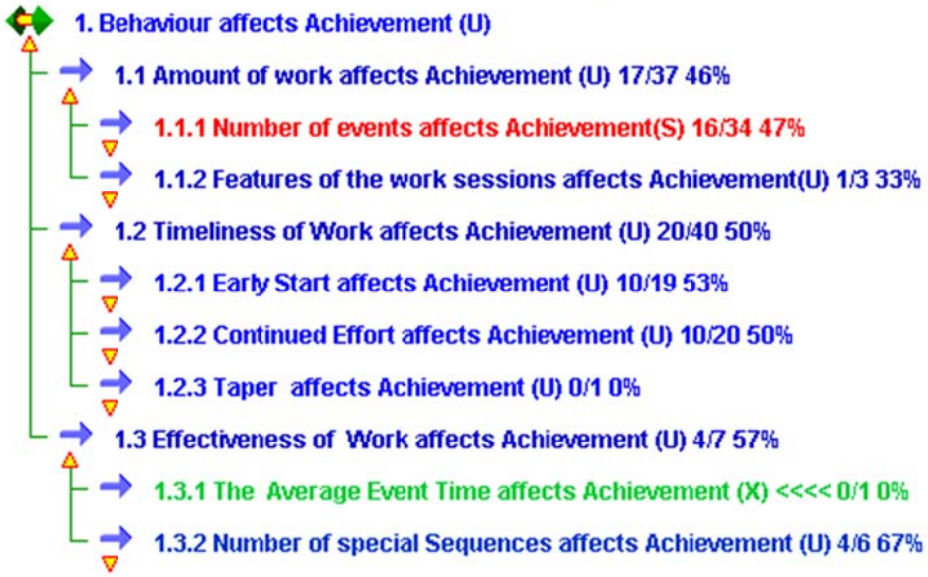


Figure 6-2. The B-A hypothesis tree at level three

6.1.1.1 The number of events (hypothesis 1.1.1)

The first group of significant hypotheses apply to the number of events over the entire semester. There are 16 hypotheses in this category, which is 47% of those tested in this subtree; they are listed in Figure 6-3.

Each hypothesis tested whether there was any difference in the achievement of students, who had been categorised according to the amount of work that they had completed during the semester. At the top level, 1.1.1, the hypothesis tested whether the total number of events affected achievement, while below this, the hypotheses related to the number of a group of related events or a specific event that was performed.

The fact that this hypothesis was found to be significant indicates a difference between students' achievement in five activity level groups from low numbers of events to high number of events. Post-hoc testing revealed a significant difference in achievement between the low activity and two highest activity level groups; so low activity levels often resulted in low achievement. The mean score of the lowest activity group was 37% (s.d. 29) which shows that many of these students failed the exam. The means of the two highest activity level groups were 66% (s.d. 21) and 67% (s.d. 23) respectively and the vast majority of students in these groups passed the exam.

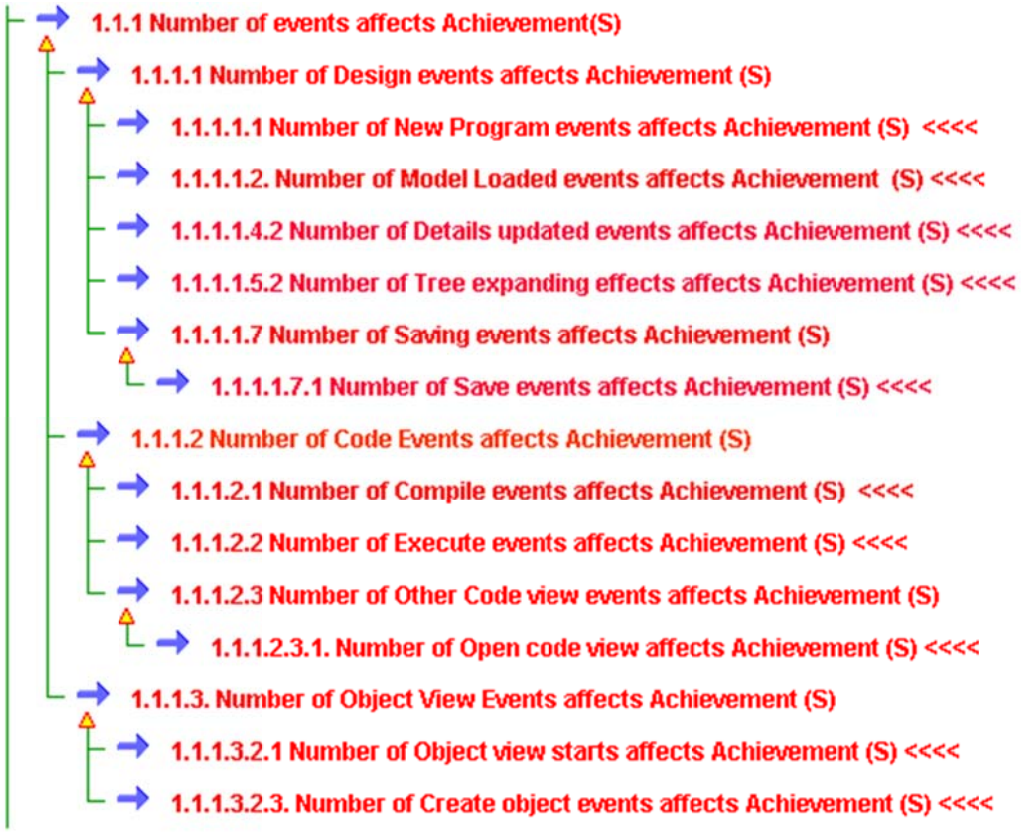


Figure 6-3. The 16 significant hypotheses that relate amount of work to achievement

It has been shown that those students who did the most work often achieved better results, and it is worthy of speculation that if students could be encouraged to do more, their results could improve. It should be noted that this research was restricted to work completed by students in the university computer labs and did not include use of P-Coder or any other resource elsewhere. It is unknown what proportion of programming was completed in the university labs or elsewhere but it may be worth investigating whether work done in the computer laboratory is more valuable than work done elsewhere. This is an issue for further research.

The remaining hypotheses in this group demonstrated that a number of different Designer, Code and Object view events and groups of events were significant when related to achievement. The ten individual significant events were:

- the number of new program events,
- the number of model loaded events,
- the number of details updated (as opposed to simply viewed)

- the number of tree expanding events
- the number of save events
- the number of compile events
- the number of execute events
- the number of code view starts
- the number of object view starts
- the number of create object events

They are mostly of interest here for their contribution to the overall number of events since each of these will occur naturally in the second grouping scheme. However it should be noted that some behaviours have been identified that affected student achievement.

6.1.1.2 Features of the work session (hypothesis 1.1.2)

These hypotheses covered various features of the work session, including total time, average time and the number of days. Just one of them was shown to be significant with respect to achievement and this was the number of days on which programming was done (hypothesis 1.1.2.3). It is not surprising that the time spent programming was not significant since the data collected reflected the total session time. The time actually applied to the task could not be determined.

There was a significant difference in the achievement of the five activity level groups depending on the number of days that they programmed in the university computer labs. Post hoc testing revealed a significant difference between the lowest group (mean of 35.0%) and highest group (mean of 67.4%), which suggests that the additional programming sessions had a positive effect on achievement.

There were fifteen scheduled laboratory classes during the semester, covering an elapsed seventeen week semester. Those students who worked on average at least once a week in the university computer labs were more likely to achieve higher results than those who worked on fewer occasions. Those students in the lowest activity level group all worked in the lab on less than 10 days.

6.1.1.3 Early start (hypothesis 1.2.1)

This section of the tree investigated the cumulative use of different groups of events at a number of different stages in the semester, specifically at the end of weeks 3, 5 and 7. The tests were whether there was a significant difference in the achievement at the end of semester given the frequency level group for each event or group of events early in the semester. Ten significant hypotheses were found in this category and they all relate to the early use of the code view, so although simply doing more programming early in the semester did not significantly affect achievement, making a greater use of the code view did.

The earliest hypotheses that were found to be significant were at week 5 and these are listed in Figure 6-4. As was expected since this data was cumulative, matching sets of hypotheses were also significant at week 7. Different results at these various stages of semester were thought to be unlikely although they were theoretically possible.

Figure 6-4. Significant Early Start hypotheses

The most general hypothesis in this group (1.2.1.1.2.2 in Figure 6-4) linked the number of code view events at week 5 to achievement at the end of semester. The difference in the amount that this view was used is striking and illustrated in Figure 6-5. It can be seen that the 20% of students in the lowest activity levels had barely used this view at this stage in the semester whilst those in the highest activity levels had made significant use of it.

These groups were formed based on the activities of students at week 5 and yet the difference in achievement at the end of semester was substantial. A student receiving the average exam score of either of the two lowest activity level groups would have failed the exam. There was a monotonic increase to the mean score as activity levels increased and it is notable that this is the case at just one third of the way through the semester.

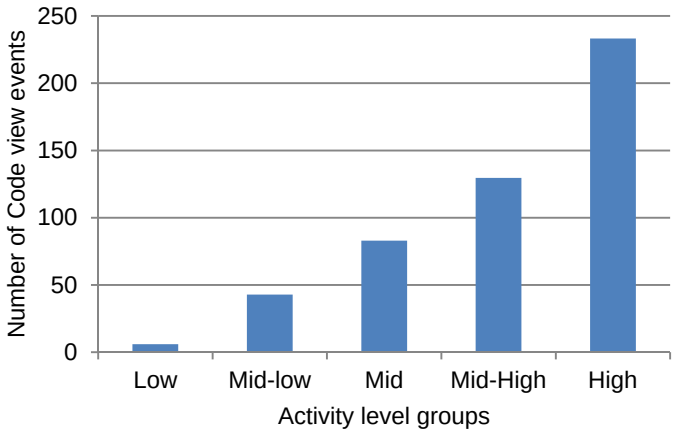


Figure 6-5. Use of Code view in activity level groups at week 5

The mean results of the two lowest activity levels (41.5% and 44.8%) and the results from the highest two activity levels (67.5% and 70.9%) suggest that it is important for students to be using the code view at this stage in the semester and any student who has not done so is at risk of failing. It may be possible to improve the overall achievement of students by identifying the students with low activity levels and offering additional assistance or encouragement while there is still be time for an intervention to be effective.

Another group of hypotheses that relate to the overall use of code view is discussed in section 6.1.2.2.

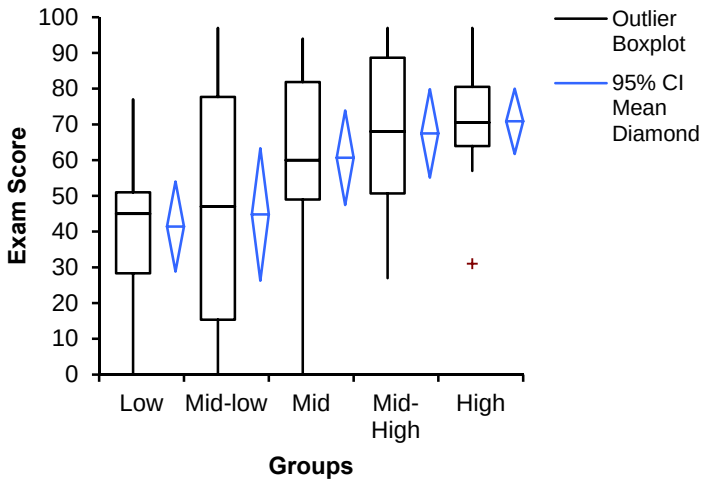


Figure 6-6. Code events by group at week 5 versus exam score

6.1.1.4 Continued Effort (hypothesis 1.2.2)

This set of hypotheses tested whether the number of events that were completed in three week blocks over the semester had an effect on

achievement. Three week blocks were selected to smooth the large variation in the number of events from week to week and still allow some differentiation through the time periods of the semester that would not have been possible if smoothing had been over longer time periods.

The ten hypotheses that were significant were those that counted events in the second and fifth tri-week block, reflecting weeks 4-6 and 13-15 of the semester.

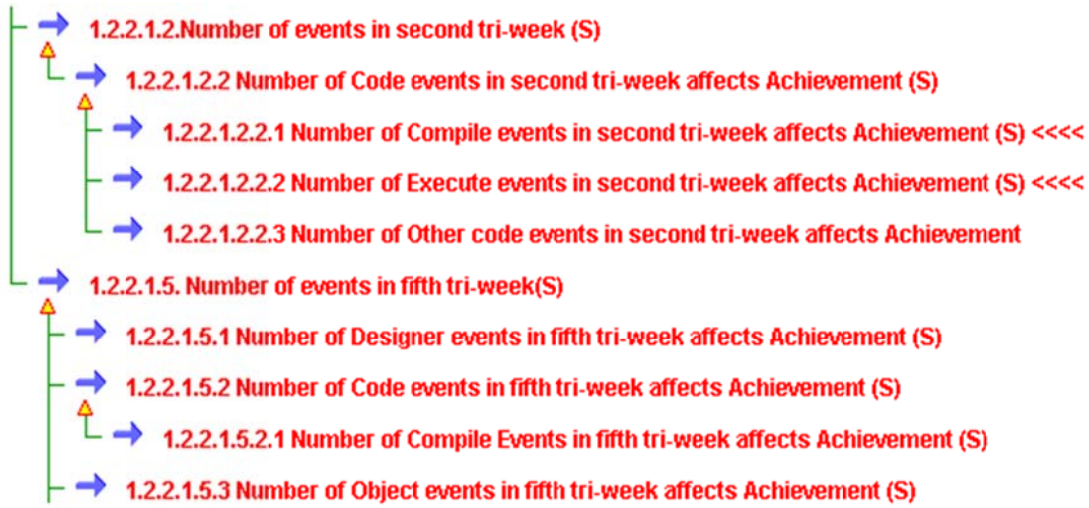


Figure 6-7. Significant hypotheses in the 1.2.2 Continued effort sub-tree

Despite the ANOVA identifying a significant result in the second tri-week, Sheffé's post hoc test did not reveal any significant differences between the groups. This possibility was raised in section 4.9 because Sheffé's test computes a new critical value for ANOVA's F value that considers the number of groups and has been identified as having a relatively high type-II error rate. In this instance there is a difference between Sheffé's and LSD post-hoc results. The LSD test on the other hand, is known to not control well for Type 1 errors and the LSD test identifies as significant four different pairs of groups, Low vs Mid-high, Low vs High, Mid-low vs High and Mid vs High.

Underlying the results for the 2nd tri-week, the only events of significance were those in the code view and the importance of using code view early in the semester was discussed in section 6.1.1.3. This is in contrast to the fifth tri-week when a significant difference was found between the lowest (mean 35.8%) and two highest activity level groups (means 70.0% and 74.4%). During the fifth tri-week both code and designer events were significant.

This is the time when students were working on their major project and this is certainly the period when they are normally expected to be most actively programming.

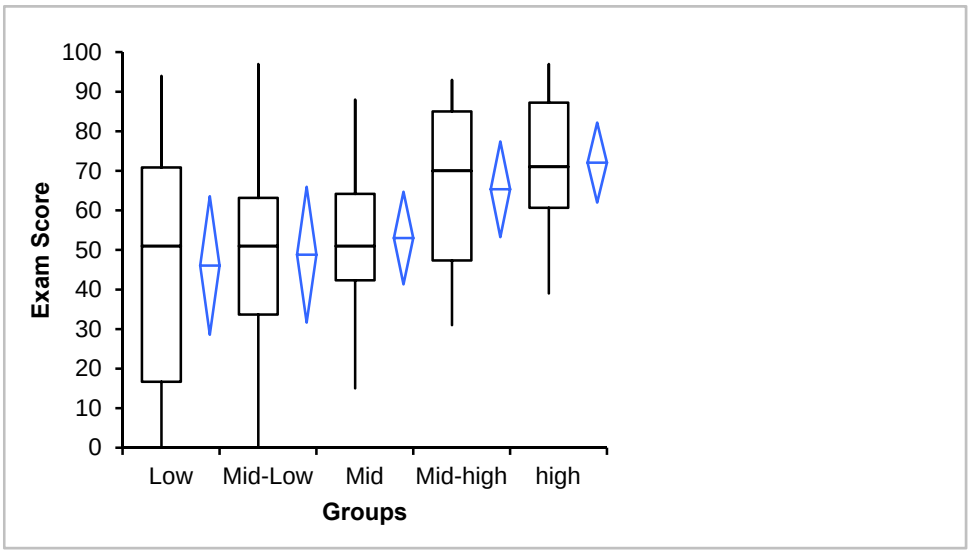


Figure 6-8. Total events in 2nd tri-week

The total number of events generated by students in three achievement levels, the lowest third, the mid third and the highest achievers are shown in Figure 6-9. Three groups were chosen for this example rather than the five that have been used elsewhere, because when five groups were used the figure was cluttered and the pattern of usage was difficult to interpret. The time period between weeks 12 and 15 shows a very large difference in the activities of the different levels of student. It is interesting that the high volume of activities by the low achievers very late (weeks 16 and 17) in the semester do not appear to contribute to achievement.

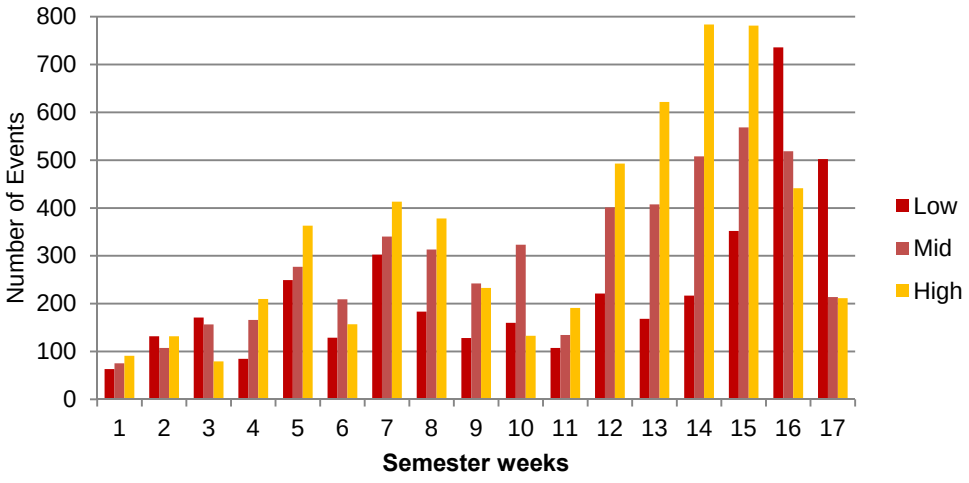


Figure 6-9. The number of events generated by three achievement groups

6.1.1.5 Special sequences (hypothesis 1.3.2)

Whatever the precise process that a programmer uses to develop a program, there are some sequences of events that will occur naturally. For example, if a program is successfully compiled then it is natural to test it and hence the compile-execute sequence would be expected to occur frequently. However some students may be prevented from using this sequence because compile errors prevent the program from being executed.

The edit-compile-edit sequence will also occur naturally during the development of a program. However since the most successful programmers might be expected to often execute their programs after a compile, the use of this sequence may not be indicative of the highest achievers.

The hypotheses in this section all tested sequences of events to discover whether they affected programming achievement. These sequences are a subset of those that are expected to be generated as a normal part of the programming process when using P-Coder. The sequences that were found to be significant were:

- Compile-execute
- Edit-compile-edit
- Multiple execute sequences
- Object sequences with method evaluate

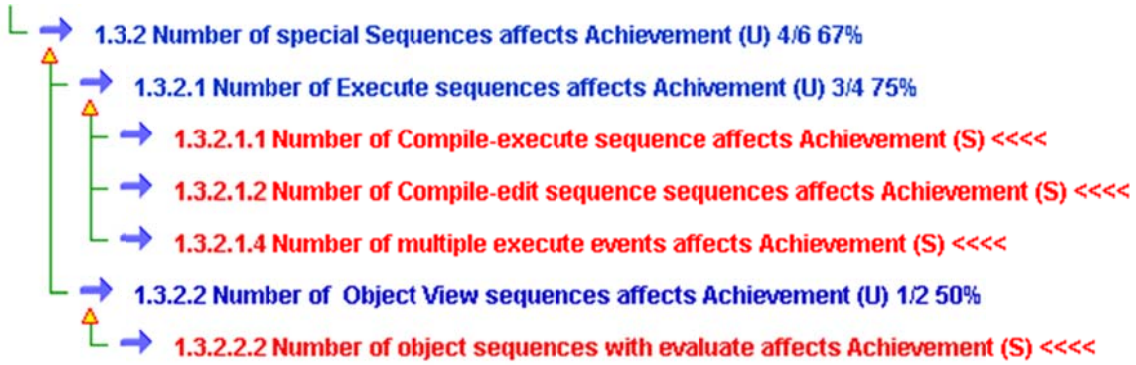


Figure 6-10. Sequences that were significant

The compile-execute sequence (1.3.2.1.1 in Figure 6-10) is a fundamental requirement to achievement in programming and would be expected of a competent programmer. It is also a sequence that may seem elusive to a beginner as he/she battles with the compiler. The five activity level groups

formed to test this hypothesis have means which are monotonic increasing (Figure 6-11). So students who generate this sequence frequently are more likely to have high levels of achievement. Post-hoc testing reported the significant difference between the lowest activity group (mean 39%) and the two highest activity groups (mean 72% and 74%). There is a 21% point increase in the means between the mid (51%) and the mid-high groups (72%). The achievement of more than 75% of students in the two highest activity level groups is higher than the mean levels in all of the remaining three groups.

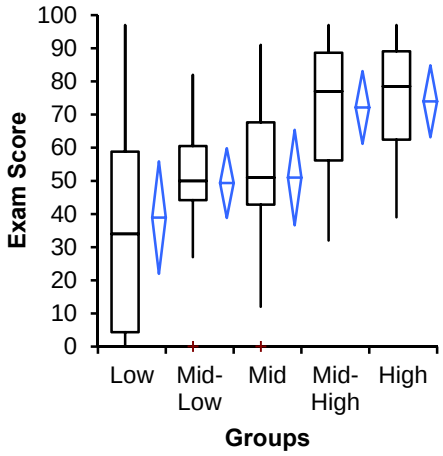


Figure 6-11. Achievement of Compile-execute frequency groups

Hypothesis *1.3.2.1.2 Number of compile-edit sequence affects achievement* investigated the compile-edit. This sequence is a normal part of the programming process. It occurs when a programmer after working on one section of code has that section compile correctly and then he/she moves on to work on a different part of the program or if a compile error occurs and they must return to editing to correct the error.

In the early days of programming (circa 1960s and early 1970s) compile errors resulted in a significant delay to progress because of the long turn around between compiles; therefore careful reading of code was essential. So this sequence would have been expected to be associated with low levels of achievement. With the advent of IDEs students have changed their use of the compilation process to include the discovery of syntax errors and this has been encouraged by the speed of the compilation process.

So this event sequence will be produced by all programmers but it may be expected to occur more frequently for those who are unable to achieve a clean compile and hence must repeatedly return to the edit process rather than being able to move on to execute the program.

Although the overall ANOVA test indicated a significant result, there was no indication of significance from a post-hoc test between the activity level groups (as in hypothesis 1.2.2.1.2 discussed in section 6.1.1.4). The mean achievement of the lowest activity group was close to 40%, while the remaining four groups all had quite similar results. So the importance of this sequence is that those students who used it very little are more likely to have poor levels of achievement.

Multiple execute sequences (1.3.2.1.4 in Figure 6-10) would be seen from a programmer who is more thoroughly testing a program rather than simply seeing if it runs. Higher numbers of multiple execute sequences were found to be indicative of higher scores and the means of the activity groups are monotonic increasing (Figure 6-12). An interesting feature from this figure is the extremely high median score of the highest activity level group of 87.5%. This shows that half of the students in the high frequency group of the multiple executes did indeed achieve particularly high scores.

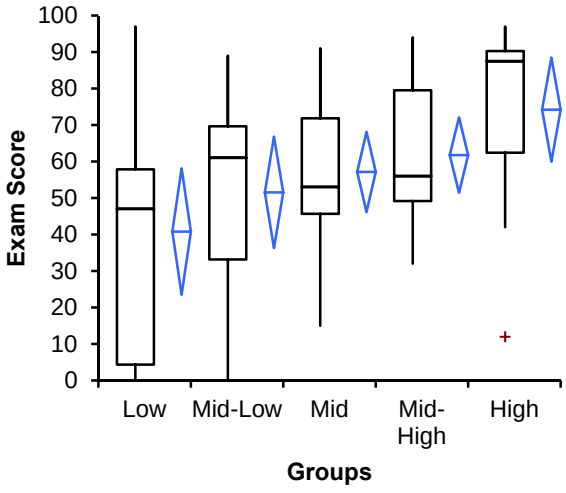


Figure 6-12. Achievement of multiple execute frequency groups

Hypothesis 1.3.2.2.2 tested whether *the number of object sequences with evaluate* affected achievement. One of the major benefits of the object view is said to be the ability to evaluate methods independently of a main program (Kölling et al., 2003). The fact that the object view sequences that

included an evaluate event were significant, when sequences that did not include an evaluate event were not, lends support to this view. The mean of the lowest activity level group was just above 30, but the middle three groups all had similar means in the 60s and that of the highest activity was lower in the 50s. Unlike the other sequences in this section, post-hoc testing identified the differences in achievement as being between the lowest activity level and the middle three groups. So although some use of this sequence does appear to be helpful, high use of this sequence does not indicate high achievement but rather may be indicative of some students who are struggling to have an entire program to execute and so continue to evaluate individual methods instead.

In the course of study under observation, the use of Object view in P-Coder was not introduced until around the middle of the semester. This result strengthens the idea that it would be useful to introduce its use earlier and ensure that all students are aware of its benefits. Use of the evaluate event is similar to the use of the execute event in that the programmer is able to see the results from a program and should therefore be able to assess the success or otherwise of the program that they have created. It may be that this is a key element to understanding the computational process.

The special sequences have revealed three different types of relationships between the attributes even amongst those that were significant.

1. Edit-compile-edit was a sequence where a deficiency was detrimental; in other words the lowest activity level group had poor achievement in comparison with the remaining groups.
2. Edit-execute and Multiple executes both were high activity – high achievement sequences. The more use of these sequences, the higher the achievement.
3. Object view with execute resulted in the mid-level activity having highest achievement. The highest activity levels indicated too much use of this sequence and this probably occurred as a result of students being unable to create the sequences in 2.

6.1.1.6 The LS-A tree result

To this point, the discussion on grouping the hypothesis according to the tree structure has only considered results from the LS-B tree; this section will consider the only other significant result that was found. The LS-A hypothesis tree produced one significant result and hence there is no grouping of results possible (Figure 6-1).

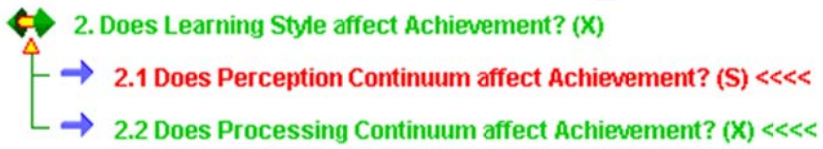


Figure 6-13. The complete hypothesis tree for Learning style - Achievement

Recall from section 3.4 that the learning style used in this research was the KLSI and this identifies the preferred learning style of students according to four quadrants along two axes that represent the perception and processing continua. This significant hypothesis tested whether the placement of students on the perception continuum affected their achievement. For this hypothesis test the students were divided into two groups by the Abstract Conceptualisation/Concrete Experience axes (normally horizontal). In the sample there are approximately three times as many students in the Abstract group as the Concrete Group.

There was a significant difference in the scores for the abstract (M=60, SD=23.1) and concrete (M= 39, SD=21) groups; $t(37) = 2.43$, $p=0.02$.

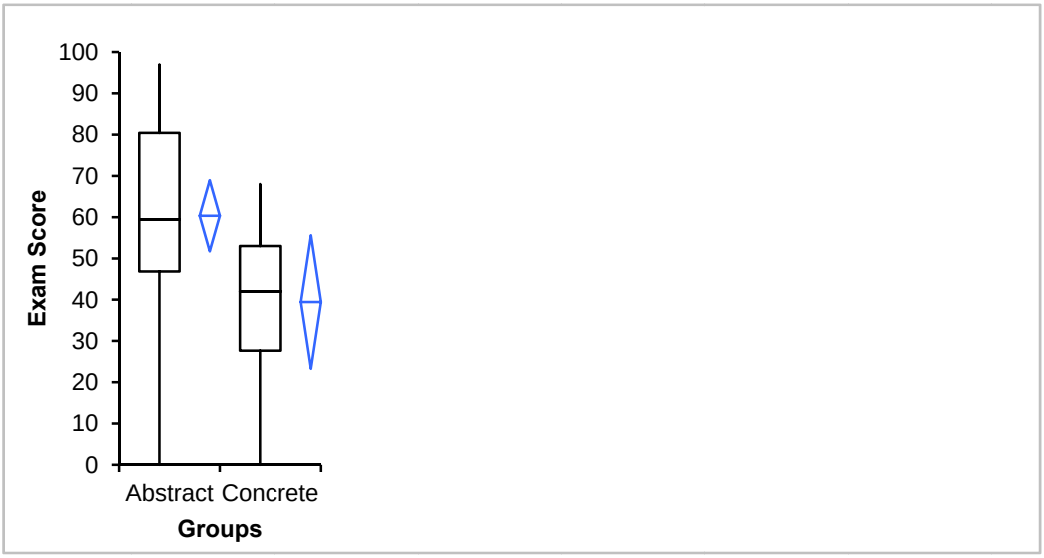


Figure 6-14. Kolb's Abstract and Concrete Groups vs. Achievement.

The grouping of significant hypotheses according to the tree structure has produced five groups from the B-A tree and a single significant result from the LS-A tree. The value of this grouping scheme will be considered in section 6.2 along with the two other schemes that follow, the tool structure (6.1.2) and the literature (6.1.3).

6.1.2 Follow the tool structure

The tool structure provides another scheme for classifying the hypotheses. The categories arising naturally from P-Coder are: Designer, Code, Object, Class and Module views. Each of these could have hypotheses allocated and categorised as in section 6.2.1

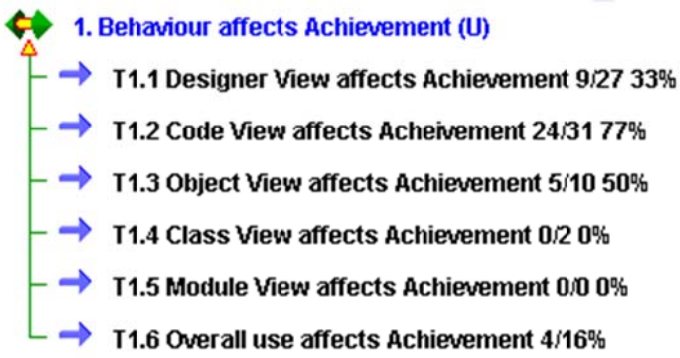


Figure 6-15. Classifying hypotheses according to the structure of the tool

The number of significant hypotheses is not identical to the previous method of classification, discussed in section 6.1.1. Classifying hypotheses using the structure of the tool does not result in similar numbers of hypotheses in the categories. If class and module view are excluded, on the grounds that there was too little use of these views to provide any valuable results, the groups relating to views that remain are two of approximately equal size and one rather small one. This classification does emphasise the importance of the use of the Code view in the achievement of students. It would appear that use of Designer view and Object view do play some part in affecting achievement but the largest contribution appears to be in Code view. This category has a larger percentage of significant hypotheses than any of the previous categories from either this scheme or that used in section 6.1.1.

It is possible that there was some particular aspect of the selection of hypotheses that gave undue emphasis to Code view in this partitioning and

this may warrant further investigation. Each group of significant hypotheses will be investigated in more detail.

6.1.2.1 Designer View (T1.1)

In P-Coder, use of the designer view is a fundamental part of the programming process. This is where programs are loaded or new ones created, where the main components of the program are manipulated and details of data and control structures fleshed out to eventually achieve a complete program.

There were nine significant hypotheses in this group and these included:

- the total number of designer view events
- the number of new program events
- model loaded events
- details updated events
- compile-edit sequences

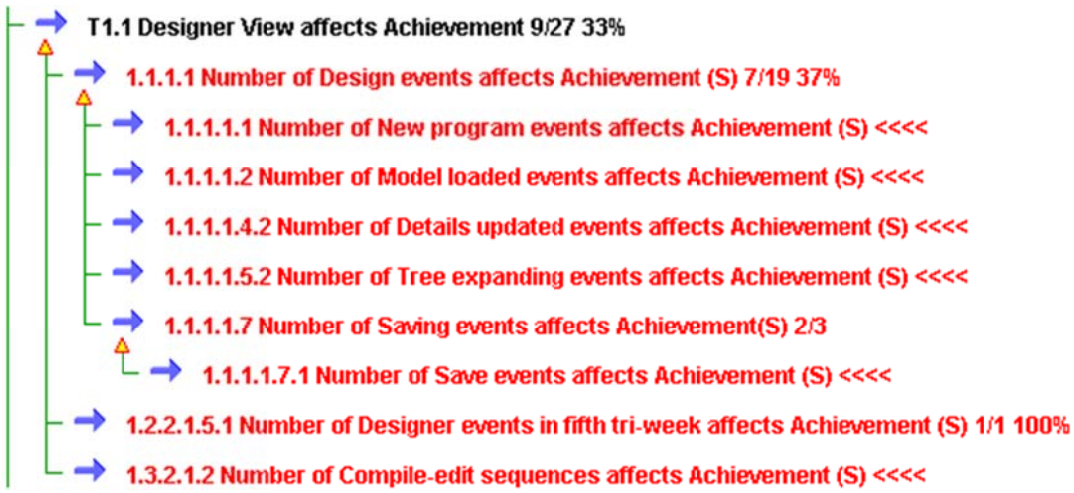


Figure 6-16. Designer view significant hypotheses

Hypothesis 1.1.1.1 investigated whether the number of designer view events affected achievement. The designer view is an essential component of P-Coder. This view is used for much of the programming process because the programmer must first assemble the algorithm and then flesh out the detail of the program using events in this view.

Post-hoc testing revealed a significant difference, almost thirty per cent difference in the mean achievement, between the highest and the lowest activity levels. The range in the number of events created was from just one

thousand up to nine thousand and in the upper half of these activity levels almost all students had a good level of achievement.

Hypothesis 1.3.2.1.2 investigated whether the number of compile-edit sequences affected achievement. In P-Coder the compile-edit sequence forces the programmer to change views from the code view to the Designer view. Compile-edit sequences are traditionally seen by programmers as an indication of failure, since they occur, in contrast to a compile-execute, as a result of not achieving a clean compile. However, another view of this sequence is that it could occur as a result of clearing one error in the program and moving on to the next section in the program. Students today, tend to use the compiler as a substitute for the proof reading of yester year. Whether it is considered success or failure is not a major concern, making mistakes (and learning from them) is immensely valuable in the programming process and the number of compile-edit sequences was found to be significant in relation to achievement.

The new program event occurs because of a menu selection to start a new program and only the highest level uses of this event escaped low scores. So although the number of new program events was found to have a significant effect on achievement (hypothesis 1.1.1.1.1), perhaps surprisingly, the lowest achievement was found in the mid-low and the mid activity level groups and there is not a significant difference between those in the lowest and highest activity levels; although there is a twenty two point difference in the means. The very high mean score, in the 80s, of the highest activity level group and the compact nature of both the diamond and candle indicates that many students in this group are high achievers. It would appear worthy of speculation that if students created more programs then this would lead to higher achievement.

The model loaded event (hypothesis 1.1.1.1.2) could be seen to be in competition with the new program event above; it is used when a programmer begins work on an existing program. Also in contrast with the new program event the highest achievement was in the mid activity level, with very low achievement at the lowest activity level and reduced mean achievement at the highest activity levels.

One interpretation of this result is that students who rarely return to work on an existing program or do not load the example programs are unlikely to succeed. The dip in mean scores in the highest activity levels when compared to those of the mid-range could be indicative of students returning repeatedly to the same program and not making headway.

Hypothesis 1.1.1.1.4.2 that tested whether the number of details updated events affects achievement was placed in this group and was also significant. This event was one of the few found below a non-significant hypothesis. The sibling relationships at this level tested similar events but where the detail of an algorithm step was viewed as opposed to updated and they were non-significant. So updating the algorithm was associated with higher levels of achievement but viewing it was not.

6.1.2.2 Code view (T1.2)

The benefits of using Code view early in the semester were discussed in 6.1.1.3 but the total use of Code view over the entire semester was also linked to achievement. Five hypotheses in this group that were shown to be significant are listed in Figure 6-17.

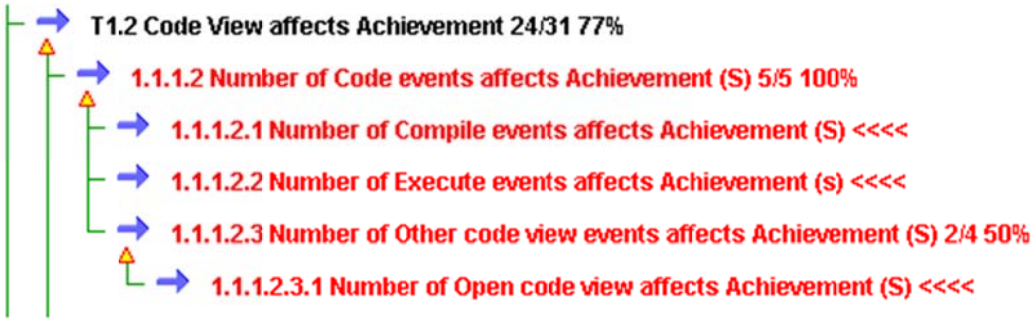


Figure 6-17. Significant hypotheses that relate the use of Code view to achievement

In contrast to the early use of code view, where the highest use was found to be the most effective, there is a more complex situation when considering the use of Code view over the entire semester (Figure 6-18). While the mean of the lowest activity group continued to have the lowest overall achievement, which at close to 40% was well below the pass level, the highest mean achievement was in the mid-high level. This indicates that although a fairly high use of Code view is worthwhile, some successful

students appear to complete their programming without many events. This finding is similar to that of Fenwick et al (2009) who found some high activity levels amongst low achieving students and described them as ‘spinning the wheels’. These highly active but not highly achieving students may also be the ‘tinkerers’ defined by Perkins, Hancock, Hobbs, Martin, & Simmons (1989). This group of students appeared to be continually changing their programs, sometimes making changes and then returning to the original program with little thought and certainly no planning.

The only code view event where the mean scores were monotonically increasing between the groups was the execute event (Hypothesis 1.1.1.2.2), this is discussed further in 6.1.3.2. The hypothesis that considered the event that opens code view 1.1.1.2.3.1 was unusual in that post hoc testing showed that the greatest difference was between the lowest activity group and the mid-level activity group, this may be an indication of high achieving students being more efficient and directed in their work.

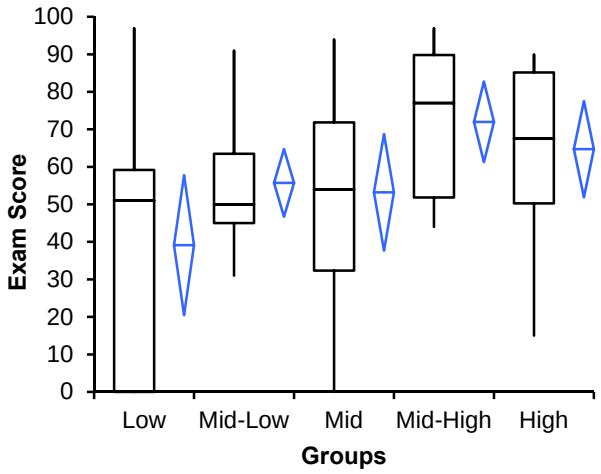


Figure 6-18. Use of Code view and achievement

6.1.2.3 Object view (T1.3)

The Object view in P-Coder allows users to handle objects independently to an executing program. It may be that novice programmers will gain a better understanding of the purpose of classes and objects and be able to run methods that will provide information on whether the program is correct, view classes, create (or load) objects, evaluate (run) methods on those

objects. The significant hypotheses that relate to the object view are shown in Figure 6-19.

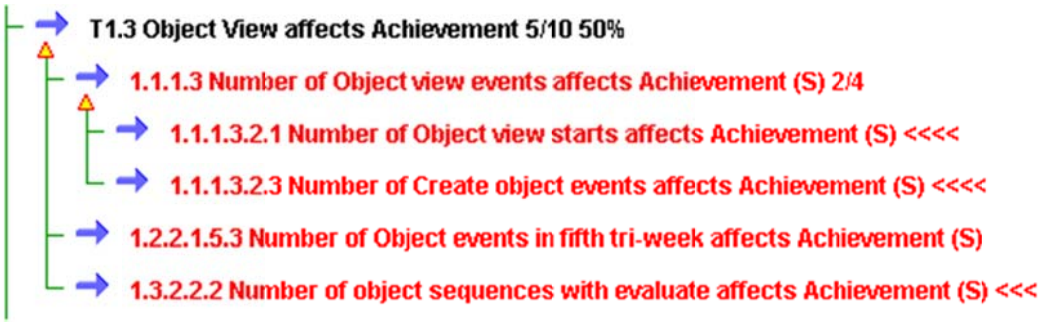


Figure 6-19. Significant hypotheses related to Object view

The total number of *Object view events* and also simply *Object view starts* was significant. When sequences of object events were considered those that included method evaluate were significant.

The fact that hypothesis *1.1.1.3.1 The number of object view starts* was significant and that the difference was found between the lowest activity group and the mid-high group shows that the use of the object view seems to be useful to students in aiding their achievement. However, the very highest levels of use of this view were not associated with the highest achieving students. It is possible that some students who are struggling for success will continue to use this view when other more successful students have moved on to complete execution of programs.

Object view sequences (hypothesis 1.3.3.1.2) are those that include several object view events and could be expected to include, at minimum, the load of an object and possibly the evaluate of a method. Only those sequences that included an evaluate event were found to be significant when associated with achievement.

The very lowest activity level group was found to be significantly different in achievement than the three mid-level groups, with mean achievement almost doubling in neighbouring groups. The highest activity level group has a mean just below that of the three middle groups perhaps indicating that very high levels of use, of this type of sequence, was not associated with the highest level of achievement.

The create object event (hypothesis 1.1.1.3.3) is one in which the programmer provides values for the constructor to instantiate objects. An

understanding of this operation is fundamental to object oriented programming so it should be no surprise that very low levels of use of this facility are associated with lower levels of achievement.

The number of create object events was found to be significant and here the three highest levels of activity had similar means, all in the sixties, with the first two activity levels having means below this level, 39 and 49 respectively.

The effect of using object view has been identified in the literature as important in aiding student understanding of OO programming (Kölling et al., 2003). The reasons given are that this provides students with a view of the program as objects (as opposed to files) and permits methods to be executed without entire programs. However no evidence is provided to support this, except for a set of guidelines for teaching that are said to be aided by the object view.

> *"The main difference between Java source interpreters and BlueJ is the level of conceptual abstraction provided by the user interface. The abstraction used or interaction in Java interpreters is lines of source code. The conceptual abstractions used in BlueJ are classes and objects, represented graphically. We believe that the initial focus on higher level concepts benefits a deeper overall understanding of object-oriented programming. The early fixation on source code can distract from important issues and hide the bigger picture. We are, however, not aware of a formal study to confirm or reject these assumptions." (p.9)*

Several hypotheses were found to support this, including:

1.1.1.3 Number of Object view events

1.1.1.3.1 Number of Object view starts

1.1.1.3.3 Number of Create object events

Of the five activity level groups (in 1.1.1.3) the mean score was highest in the group of students with the second highest usage of the object view (Figure 6-20). This supports the notion that although the Object view is useful in gaining insight, it may not be the end of the line in student

learning. Students appear to need to move on to executing free standing programs and those students that continue to work in object view may be still trying to achieve comprehension.

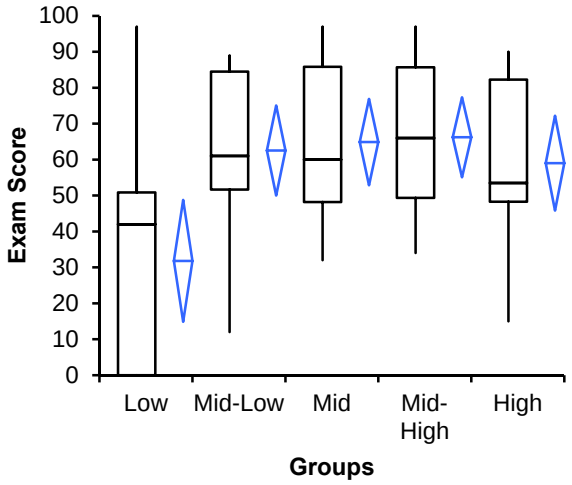


Figure 6-20. Object sequence (with evaluate) activity level groups versus achievement

Two types of sequences were investigated within object view, those that included an evaluate event and those that did not. The former were shown to be significant (hypothesis 1.3.2.2.2) whereas the latter were not. It is noticeable the lowest activity levels in this sequence had very low scores when compared with those students who used the sequence just a little more. Post-hoc testing found a significant difference between the low activity group and the three middle activity levels. This would appear to support the use of object view and especially the sequence of events that instantiates an object and executes a method; not doing this appears to be detrimental to student achievement.

There is little to differentiate the other groups, except for a slight drop in median and mean in the highest activity level. This indicates that the students that make the highest use of this sequence are not the highest achievers.

6.1.2.4 Overall use (T1.6)

There were four significant hypotheses that did not relate to the specific use of a particular view (Figure 6-21). These were the total number of events which has been discussed at length in section 6.1.1.1, the number of

sessions discussed in 6.1.1.2 and two hypothesis that relate to the timing of events through continued effort covered in 6.1.1.4

So this category provides no additional value to the discussion on classifying significant hypotheses.

Figure 6-21. Significant hypotheses relating to overall use

6.1.3 Derive from the literature

This section aims to classify the hypotheses according to aspects of programming behaviour that have been identified in the Computer Science Education (CSE) literature as important to success in programming. This scheme is important to this research because it is invariant to both the research model and tool, unlike both of the other classification methods.

Researchers have suggested that certain programming behaviour is an indication of expertise or will assist learning and the significant hypotheses will be grouped to assess the importance of these aspects of behaviour. Undoubtedly the literature used in the section will be a subset of that which is available but if any additional support can be found amongst the significant hypotheses for any of these behaviours then at the least this would indicate further investigation may be warranted.

- Design Programs

  According to Petre (1990) experts spend more time on planning and evaluating their code than other programmers. It has often been suggested that novice programmers should do more planning of their programs.

- Execute Programs

  Experts execute their programs frequently (Gugerty & Olson, 1986). Although it is extremely unlikely that there are any experts amongst students, it is possible that a student who executes their programs more often than others is able to develop a better understanding of the computational process.

- Use Object view (Kollings)

  Kölling and Rosenberg (1996b) argued that an object oriented program development environment would be beneficial to beginning programmers. P-Coder provides the object view for the creation and interrogation of objects. So it may be enlightening to look at whether use of this view does in fact contribute to student learning.

- Motivation

  Student motivation has been identified as a possible predictor of success (Jenkins & Davy, 2002; Biggs & Tang, 2007). Whilst motivation itself may not be directly measured it seems reasonable to suppose that motivated students will do more work and attend on more occasions that those who are less motivated.

- Timeliness

  Timeliness is considered important by many university teachers. This was identified in the literature by Fenwick et al (2009) who found that students who started working on an assignment early tended to score higher grades than those who started later. This research does not differentiate work done on assignments and projects to working on class exercises in workshops. So there is no additional discussion on this topic other than that presented in 6.1.1.

The significance ratio and density percentages, at a node in the tree, show the ratio of significant hypotheses to total hypotheses and the equivalent percentage of the particular sub-tree. When using the literature to classify the hypotheses, these measures were calculated exactly as before by a simple count of the significant and total number of relevant hypotheses (Figure 6-22).

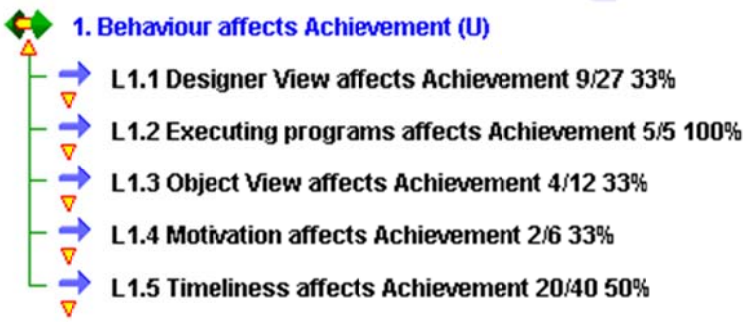


Figure 6-22. Grouping using CSE literature

6.1.3.1 Design Program (REF 6.2.2.1)

One of the main objectives of P-Coder is to make design an integral part of writing programs. It provides facilities to explicitly design the program rather than writing code as a first step. However, the use of design view does not necessarily equate to the total amount of planning done, since alternatives such as use of pencil and paper could be used by some students. The most viable option available is to consider the use of design view which is identical to the tree level T1.1 using the tool views as a group mechanism (Figure 6-15).

6.1.3.2 Execute programs

The number of times programs were executed was significant, as a total number of events for the semester and as early as week 5. The number of multiple executes was also significant along with the number of object sequences including an evaluate method.

These are all indicators that executing programs is linked to achievement. This may be because an executing program is able to produce output that, whether as expected or not, aids understanding of the computational model.

It is also likely to be the case that more successful students will achieve clean compiles sooner and reach the position of being able to execute their programs sooner but this would not necessarily indicate an increased total number of compiles. Object sequences with evaluate (Hypothesis 1.3.2.2.2), were also significant. These sequences included running an evaluate method, this is testing an individual method in a classe rather than an entire program.

Cognitive constructivism suggests that students create their new understanding from new experiences combined with what they already know (Fosnot, 1996). It is clear that when a program executes students get feedback and it appears that they are then able to internalize this.

The compile – execute sequence (hypothesis 1.3.3.1) is one that might be expected to be important. To some degree, a clean compile is an indicator of success, in that a clean compile permits testing by the execution of a programs. There have historically been some changes to the view of the purpose of compilation of programs. In earlier days, an error in compile meant wasting large amounts of time in the turn around to resubmit. Whereas today, compiles are very commonly used to locate trivial syntax errors. A clean compile permits testing and it has been found that expert programmer execute their programs more often.

The Compile – execute sequence was found to be significant, with the Low activity level group achieving significantly lower results than both the highest two activity levels, with a 35% difference in the mean of the lowest and highest activity groups. The mean of the groups was monotonically increasing so that there appeared to be no penalty from the highest activity of this sequence. 75% of the lowest activity level group achieved less than 59% and 75% of the highest activity level achieved more than 62%.

Multiple executions of the same program (hypothesis 1.3.3.1.3) are likely to be an indicator that the programmer is testing their program thoroughly. The achievement of students in different activity levels of this sequence of events was found to be significantly different. The means of each group were monotonic increasing with the highest mean achievement in the highest activity level group.

6.1.3.3 Manipulate Objects (Ref 6.1.2.3)

This category is the same as T1.3 in the tool category (Figure 6-15) and was discussed in section 6.1.2.3.

6.1.3.4 Motivation

Motivation has not been directly measured in this research but a number of significant hypotheses were found that can be meaningfully placed in this category. They include the amount of time spent, the number of occasions when working in the labs and the total amount of work done.

Interestingly the amount of time spent using the software in the lab did not have a significant effect on achievement. This may be because students were in the lab and using the software but not working effectively, possibly because they were attempting multi-tasking or simply that total time spent is not related to achievement.

The number of days spent programming did have a significant effect on achievement. This may have been strengthened because of a link to other issues such as the number of lectures (not laboratory classes) attended but this information is not available.

The largest impact on achievement is in the group of students with the lowest levels of attendance, and their mean of 35.3% indicates very poor results (Figure 6-23). The low median score (51%) in the low-mid group demonstrates a cluster of students in this group who are barely, if at all, passing the exam. It seems likely that these students would benefit from an increased number of sessions.

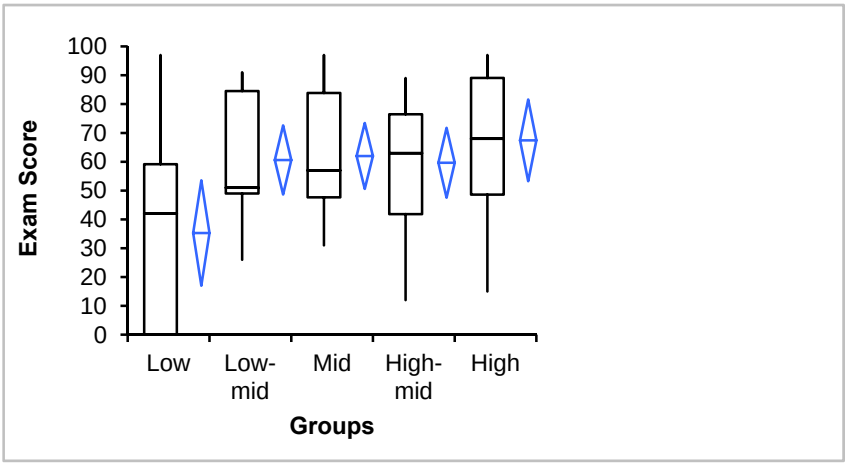


Figure 6-23. Number of programming sessions versus achievement

6.2 The value of the three classification schemes

This section investigates the results from the three classification schemes to appraise the value of the model. A comparison of the results across the three schemes will help to determine to what degree there is any commonality between them. A consistency amongst the results would signify usefulness and indicate robustness of the research model whereas fundamental differences between the results would be indicative of fragility in the model.

The three classification schemes that have been used to categorise the significant factors are illustrated in Figure 6-24. The diagram shows the density of significant hypotheses within each of the categories as identified by the three schemes, the tree, the tool and the literature. To compare the schemes and discover whether there is any commonality between them, the factors are separated into three bands:

- High (Red border) : Those with a high density (67% or more) of significant hypotheses
- Medium (Purple border) : Those with medium density (less than 67% and more than 33%) of significant hypotheses
- Low (Blue border): Those with a low density (33% or less) of significant hypotheses

Each of the three classification schemes produced one category where the density (percentage) of significant hypotheses was considerably higher than the other categories. When using the tree structure this was *Special sequences*, for the tool structure this was the use of *Code view* and for the literature grouping the group of hypotheses labelled *Execute*, that relate to executing programs. This final category stands alone as the only one where 100% of hypotheses were significant.

A more detailed examination of these noteworthy categories reveals that the *Execute* group consists of five hypotheses and is essentially a subset of the larger group (24) of *Code view* hypotheses with the exception of a single hypothesis from the *Object view* group. The *Special sequences* group are mainly sequences that include compile and execute sequences, which cover both the use of *Code view* group and *Execute* group.

The use of the tool structure to classify the hypotheses emphasises the importance of Code view to achievement and this could be thought to be overlooked in the tree structure. However, Code view includes the very important operations of compile and execute, that were part of the Special sequences. The hypotheses that investigated the worth of code view were spread across several subtrees and given a greater emphasis by this grouping.

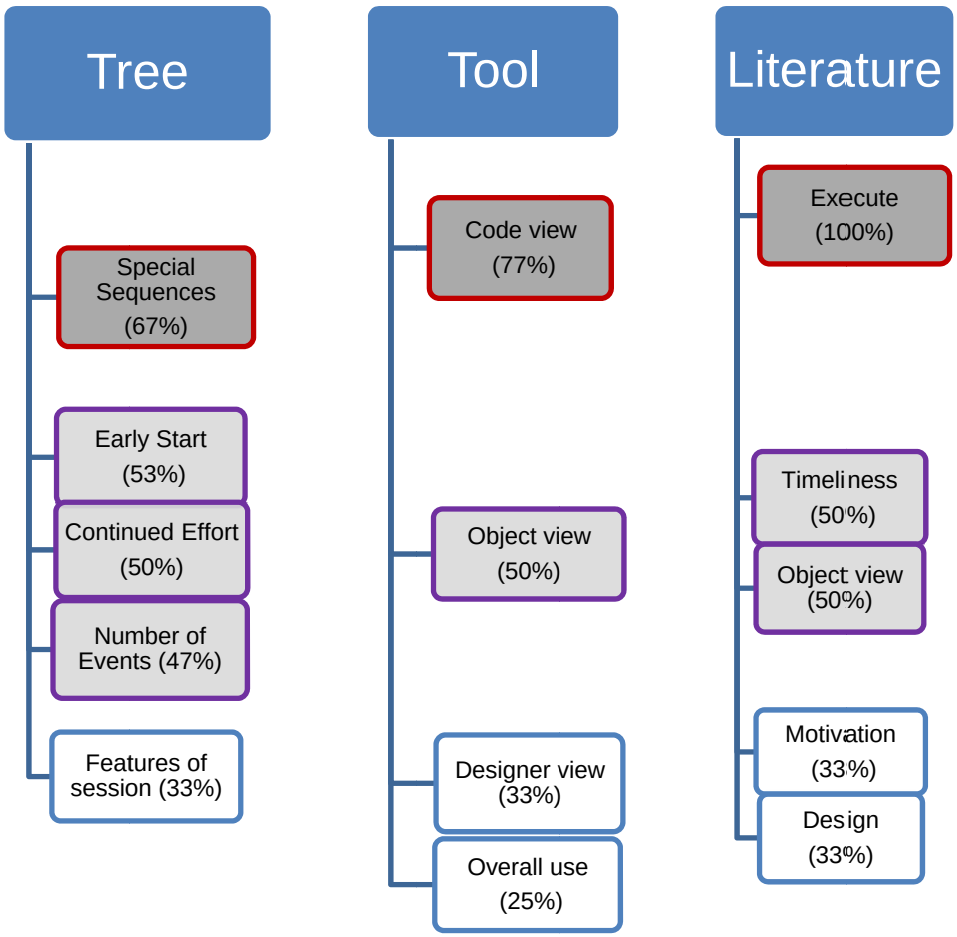


Figure 6-24. The factors within the classification schemes at three density levels

The different branches of the tree included:

- the overall use of code view,
- the early use of code view
- important programming sequences.

This category of the significant hypotheses appears to indicate that this view has critical importance to the achievement of students. It appears that use of code view assists in understanding and although many compile events is

indicative of high achievement, many times executing programs is even more so.

Using the hypothesis tree as a classification scheme identified several different areas of significance to achievement. The special sequences group was smaller by count because few sequences were defined and tested but a high percentage of them were significant. The three stronger groups, identified both by count and ratio, in the hypothesis tree were Early Start, Continued Effort and the Number of events. These groups can be associated with the timeliness group from the Literature.

The contribution of using the literature to classify the significant hypotheses is to include executing programs and timeliness. The literature specifically identified the fact that expert programmers execute programs often and this appears to be important to the progress of novices as well. Timeliness was very similar to the Early Start group from the Tree grouping.

The other categories identified from the literature had already been recognised in the previous sections except for motivation. This is a complex issue and attendance, which was shown to be significant, may be related to it. However, there are many possible facets that could be measured and only a few of which have been recorded in this research.

It is only possible to speculate here on why the execute method is so important and exploring the reasons for this is an area for future research. It was discussed in section 6.1.3.2 that the literature has shown that expert programmers executed their programs more than journeymen and executing programs appears to be important to the learning of novices. It may be that since learning to program requires students to understand the nature of the computational process, seeing the outcomes of their own programs is particularly valuable.

The literature noted that timeliness in starting assignments could be related to achievement but this research did not differentiate between programming on exercises or assignments. So an issue for future research to investigate if one type of programming is more valuable than the other. Overall, the stages of programming are mirrored in the density of hypotheses. It appears that compiling and executing programs have a critical importance in

developing the understanding of students. This will be discussed further in section 7.3.

6.3 A review of the model

This section will review the model and its worth as applied in this research.

The model is repeated in Figure 6-25.

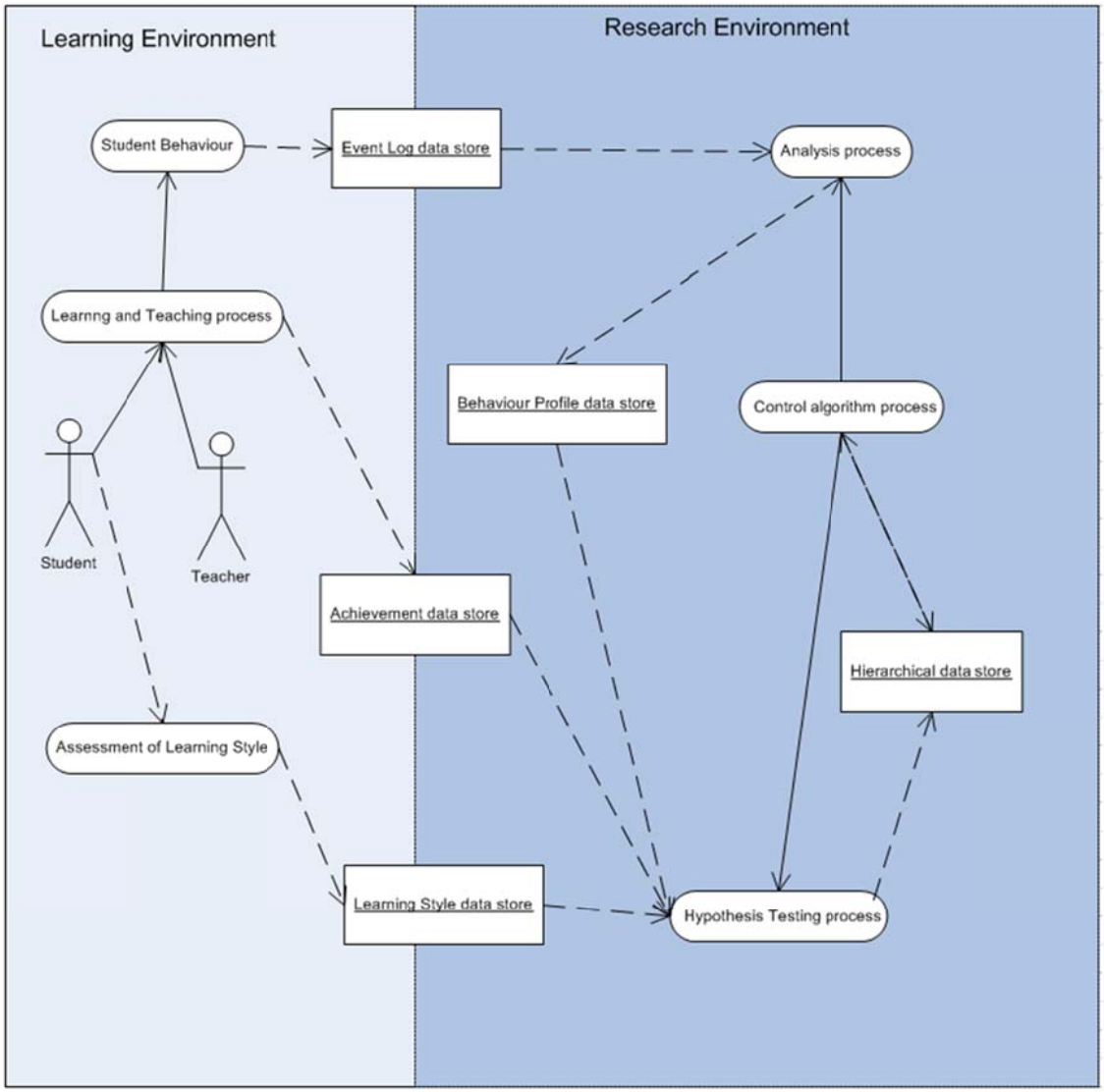


Figure 6-25. The structure of the research model (copy of Fig 4.1)

The focus of the discussion in this section is on the operation of the Control algorithm and its relationship to the hypothesis testing process, as it is the most important part of the model. Recall that operational issues with the Control algorithm were discussed in section 5.8 so the discussion here is at a higher level.

There were two significant issues that arose in relation to the disaggregation and exploration of data, which was the major objective of the model. One of these surfaced during the design of the control algorithm and the other during the application of the model. During the design of the control algorithm, the issue was, at what point the control algorithm should direct the disaggregation of attributes to cease. Whilst during application of the algorithm the main issue was selecting the child attributes to disaggregate the attributes.

A discussion in section 5.8.6 described the five significant hypotheses that were uncovered by continuing to disaggregate data and take the hypothesis testing to a second level of insignificance. Two of these were further disaggregated to find a cluster of significant hypotheses. This occurred under the direction of the control algorithm. It clearly remains undiscovered whether there are interesting relationships hidden in the unexplored depths any of the hypothesis trees. Although, it is true that some may have been found if the pruning was delayed to three or more levels of non-significant results, there were only 13 nodes (8.1%) where pruning occurred and there was data available to continue. Given the few significant hypothesis uncovered at the second level, to continue to a greater depth would have been very difficult to justify. There is clearly a trade-off between terminating the search early when non-significant hypothesis are reached but when interesting connection in the data may remain undiscovered or continuing the search to a deeper level.

The process of disaggregating the attributes to form child hypothesis required domain knowledge to identify suitable child attributes. For example very early in the development on the B-A tree, disaggregating the "number of events" required the application of both domain and model knowledge to locate useful child attributes. The children selected here,

related to the use of the tool rather than time. Since, it was known that time was to be considered in another branch of the tree.

It is useful to ascertain whether the decisions that were made at crucial times in the application of the model had a considerable bearing on the results that were reached. If not this indicates an integrity and robustness of the model.

6.4 Summary

This chapter has grouped the significant hypothesis into categories according to three different schemes. Each one provided a different perspective in which to view the results and yet these perspectives supported each other and had a good deal of overlap.

The findings from the three grouping schemes show some similarity, it is possible to identify an underlying theme in the density of significant hypotheses and hence the overall results. This means that the model has proved itself to be robust despite the operational issues that arose in the application of the model (see section 5.8).

The next chapter will conclude the thesis by summarising the contribution of the research, noting some suggestions for future research and finally making some speculations on learning and teaching that are drawn from these results.

CHAPTER 7 REFLECTIONS AND FUTURE RESEARCH

This chapter begins by providing a summary of the major findings of the research. Then it will reflect on the research in order to consider what could be done better with hindsight.

Suggestions for related future research are provided at four increasing levels of abstraction and finally some speculations are presented on learning and teaching.

7.1 Summary of the major findings

The major findings of the thesis will be discussed with reference to the research questions. They were listed in section 1.3 and are restated here.

RQ1. Can a model be constructed to systematically test relationships between learning style, behaviour and achievement?

RQ2. Does learning style affect achievement?

RQ3. Does student behaviour affect achievement?

RQ4. Does learning style affect behaviour?

RQ1 has been comprehensively answered by creating a method to systematically identify hypotheses to comprehensively analyse data (Chapter 4). It was applied to a pertinent data set that mapped behaviour, learning style and achievement in novice programmers. During this process, weaknesses in the method were discovered and refinements were made (Chapter 5). The outcome of the application of the model were analysed and discussed to complete the process (Chapter 6), thus demonstrating that the research model was viable.

RQs 2-4 were addressed by the application of the model in Chapter 5 and the findings presented in section 5.7. The model applied to each question

resulted, respectively, in the LS-A tree (section 5.7.1), the LS-B tree (section 5.7.2) and the B-A tree (section 5.7.3).

The model

The model was used to direct the research and, although it was not fully automated, it provided a systematic process. The guidance that the model provides, regarding pruning the hypothesis tree, avoided combinatorial explosion and has produced a practical solution.

The initial decisions around the primary hypotheses and the nature of the statistical tests were the most challenging part of applying the model but after this the control algorithm indicated which hypotheses were to be tested and specified how the results should be classified in the hypothesis trees. Domain knowledge was required to select appropriate child attributes but once selected, the additional hypothesis were routinely generated and the research process continued steadily.

The pruning of the hypothesis tree occurred under the direction of the control algorithm and was done after finding non-significant results (accepting the null hypothesis) for two consecutive levels in the tree. This stopping condition which was built into the control algorithm led to more tests than if stopping had occurred at the first non-significant result, but also uncovered some significant hypotheses that would otherwise not have been tested. It was already demonstrated (section 5.8.5) that this was an appropriate choice for this research.

If the model was used with other datasets then this feature could be tuned to either prune the tree immediately a non-significant result was found or alternatively to delay pruning to three or more levels of non-significant hypothesis results, if it was considered important to continue delving deeper into the data.

In deciding whether this model was appropriate for another research project it would be important to give careful consideration to both the collection of data and the nature of the attributes to be investigated.

The collection of data

This research method was enhanced by having access to a rich multidimensional set of data. The use of automatically collected behaviour

data was very beneficial from two important perspectives. First, this was resource efficient because once the data collection process was in place, there was limited intervention required. A program was written to validate, extract and sort the data which was a relatively easy method of collecting data. The use of event data (discussed in section 3.6), rather than finer grained data such as keystrokes, ensured that it was useful to answer the research questions. Second, the data collection was done in the background and was non-invasive to the subjects.

Nature of the attributes

The method could also be applied to other situations that required a thorough quantitative examination of the relationship between two mapped attributes as long as each of the attributes has an appropriate measure and data can be collected.

The application to the three attributes in this research suggests that if both attributes are *n*-dimensional then the resulting number of hypotheses may be very large due to combinatorial explosion. However the application of the control algorithm ameliorates this problem especially if non-significant results occur.

Originality and Significance

In the domain of novices learning to program, few studies have approached the process of data analysis in a systematic way. The proposed research model provides an innovative and methodical approach to interrogating the data. It probes deeply into the data, yet, also defines the means to avoid combinatorial explosion should that data be multi-dimensional. This new approach has been constructed on a detailed knowledge base, which included a record of student behaviours in an authentic learning situation, and the application of models of learning behaviour.

The proposed model incorporates a formal approach to hypothesis generation and testing that has been used to derive and identify interesting hypotheses that connect the student's behaviour with their achievement in the learning tasks.

7.2 Reflections

At this point in the thesis it is useful to reflect on the research, giving consideration especially to the research process in order to define, with hindsight, what could have been done differently.

The model was applied to a single data set and while the strength of the findings cannot be taken as proof of a relationship, the significant hypotheses were classified (in Chapter 6) to seek the most important issues that had arisen. Within the limitations of the study (a small sample size and a single data set) there is some evidence that speculations can be made with respect to the learning and teaching process. These will be discussed further in section 7.3.1.

It would have been helpful to validate the model with a second data set. Unfortunately it was not possible to split the existing data because of the relatively small number of subjects and it was not possible to collect more because the course was no longer run. Hence, the research attempted to make the best use of the available data.

Despite the fact that learning style theorists claim that people with different learning styles will approach learning differently, no evidence was found that students with different learning styles exhibit dissimilar behaviour. Perhaps any differences were too subtle to be found or there are other issues that have a greater effect.

Evaluation of the work value in university laboratories

This research used a data set that was exclusively collected whilst students were programming in the university computer laboratories. Hence, the value of the work done in the labs was considered but work done at home was excluded. Remote collection techniques such as batch updating or online recovery could be used to record this additional data and this would allow several different sets of data to be considered, including a universal set (inside university labs and outside) and also both of these individually.

Although it may be technically possible to collect data remotely, and it is tempting to propose because of the potential to increase the volume of data

collected, there are both ethical and practical issues involved regarding the privacy of the students and the completeness of the data.

Perhaps it is more efficacious to work in the laboratory because of the availability of teachers and/or fellow students who may be able to assist with problems shortly after they arise, in contrast to the student who at home may struggle on or simply give up. It was noted by Perkins et al. (1989) that tutors were able to encourage students to delve deeper into a problem, whereas without the tutors probing, students may have abandoned their work. Judicious questions and suggestions enabled students to follow the thread and maintain focus. There may simply be fewer distractions in the computer laboratory than elsewhere.

Several times during this project, the researcher has been interested in the previous programming experience of the students. In fact a pre-test was performed but there was no use made in the model of pre-test data that was collected.

The one dimensional measure of achievement used in this research was that of examination score. The reasons for the selection of this measure were clarified in section 4.3.1 which explained the structure of the *Learning Environment*. However, it may have been informative to disaggregate the score into sections and further to classify the precise nature of the learning that was being tested by the examination. This could be used in a new and expanded B-A tree, although there is clearly a potential problem with combinatorial explosion.

7.3 Future research

This research has developed and applied a model that provides a systematic method of identifying hypotheses to comprehensively investigate relationships in data. Since the model is original, it provides for many potential avenues of research. These are presented in the following sections at four levels of abstraction; beginning with relatively minor adjustments to the model, the mid-levels allowing for use of different datasets and building up to changes that permit substantial flexibility in the model. Each level of

abstraction increases the number of domains to which the model could be applied.

7.3.1 Abstraction level 1

At the first level of abstraction the adjustments to the model are relatively minor. Modifications at this level include running the model with the existing parameters but with another set of students at another institution. Within the model itself, the stopping condition in the CAP could be altered to increase the number of insignificant levels before halting expansion. Increasing this above the existing two, could result in a lot more work for very little return but it could also uncover interesting results that have been hidden. Decreasing the number of levels can be done trivially by pruning the tree.

Adjustments could also be done at other points in the application of the model, such as in the application of statistics. The ANOVA could have been used with three or seven groups instead of five and the probability level selected for α might be altered.

Each of these adjustments would likely result in a different number of hypotheses being found significant but they can all be considered to be tuning of the model.

7.3.2 Abstraction level 2

At the second level of abstraction some more significant changes could be made. The variables that were used could be replaced by other similar ones. For example, a different learning style paradigm or a different model of achievement or a different learning tool could all be substituted into the model in a straight forward manner.

It would also be possible to more comprehensively explore the effect of student behaviour on achievement by including another data collection mechanism that would allow consideration of other aspects of student behaviour. This could be another automated recording such as recording the screen which would give an insight into other activities (such as email and social networking) that students are doing online. A video recording of the lab would provide an additional view of student activities. The requirement

to deal with the volume of this data provided by this type of investigation may make it more appropriate to use a case study approach.

7.3.3 Abstraction level 3

Further changes could be made to the model at a higher level of abstraction by changing some of the variables under investigation. An example in the same domain could apply motivation against behaviour and also against achievement. It would also be possible to select variables from other domains especially those where an automated log of behaviour could be kept.

7.3.4 Abstraction level 4

The highest level of abstraction considered here could change the number of entities that are being tested. The model currently is seeded by hypotheses that have one independent variable and one dependent variable. A one-way ANOVA was applied to test whether a hypothesis was significant. This could be altered to run the model with two independent variables and one dependent variable and use a two-way ANOVA.

The mechanism used to disaggregate all three variables and delve into the data would clearly require a careful definition and could result in a large tree, depending on the dimensionality of the data. However this would not preclude the application of the model. It is possible that with other statistical methods the number of variables could be increased still further.

7.3.5 Other

It is common in many applications of trees and graphs that weights are applied to nodes and/or arcs. Such techniques provide for further application of the data or results and may strengthen the application of the model. Future research could investigate whether it was appropriate or useful to apply weights to the hypotheses. The weight could perhaps be used to rank the hypotheses (e.g. If the literature indicates that code views are more important).

This thesis has generated a number of significant hypotheses that have been shown a relationship between student behaviour and achievement; each of these could provide avenues for future research.

7.3.6 Summary of future research

A large number of avenues for future research have been identified. The model can be altered at four different levels of abstraction that have been described. There are also additional research projects to be found in the significant hypothesis uncovered in the application of the model and some of these are discussed in the next section on learning and teaching.

7.4 Speculations on learning and teaching

This research has not collected any data specifically about teaching processes so it is unable to make any recommendations for a change in teaching processes to adopt methods that could improve learning outcomes.

This research has shown that students who exhibited different quantities of certain behaviours achieved different results. It is possible to speculate that learning outcomes for some students could be improved by adopting teaching processes that encourage the most constructive of these behaviours.

Timing

It is a part of university folklore that students are expert at procrastination. This research has shown that students who do not participate fully in programming early in the semester are more likely to fail. A group of students at risk was identifiable as early as week five.

If the behaviour of students was monitored during the semester it would be possible to identify students at risk, early in the semester and offer additional assistance to them, whilst there was still an opportunity to advance their learning. Of course, the astute teacher with small classes will do this informally anyway but more commonly classes are large and possibly remote and in these situations the provision of this information would be helpful.

Attending class

Significant hypotheses showed that students who did more work were more likely to have higher achievement. So it could be argued that if students could be encouraged to do more work, they might be expected to improve their learning outcomes. One significant attribute that contributed to more work was the number of days on which they did programming so if ways could be found to encourage students to attend classes then achievement may improve.

Emphasise testing

Students in the higher activity level groups in terms of executing programs had higher levels of achievement than those who executed their programs less. One way of encouraging students to execute their programs more would be to require them to create and follow a test plan for each program. Scaffolded examples of partially completed code may also be helpful in this regard because smaller chunks of code would need to be written before a program was in a form that could be executed.

Encourage reading of programs

When learning a new natural language it is well recognised that understanding is easier than producing language. This principle has also been studied in programming (Fuller, Johnson, Ahoniemi, Hernan-Losada, Jackova, Lahtinen, Lewis, McGee Thompson, Riedesel, & Thompson, 2007). It is quite conceivable that by reading and testing programs that a greater understanding of the computational process could be gained in parallel with assimilating the language syntax.

Provide Scaffolding

One significant impediment to executing the programs for novice programmers is not achieving clean compiles. Scaffolded examples could encourage particular programming behaviour, which may include use of the object level and partly completed programs

7.5 Concluding remarks

This thesis was about creating a research method that would facilitate the comprehensive exploration of data. It has demonstrated the outcomes that can be achieved from the application of the method.

Although the model was not totally automated, it has provided a means by which potential combinatorial explosion can be reduced in a relatively thorough search for important relationships in the data.

REFERENCES

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In M. F. Usama, P.-S. Gregory, S. Padhraic & U. Ramasamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 307-328): American Association for Artificial Intelligence.

Allert, J. (2004). *Learning style and factors contributing to success in an introductory computer science course.* Proceedings of the IEEE International Conference on Advanced Learning Technologies, Joensuu, Finland.

Atchison, W. F., Conte, S., D. , Hamblen, J. W., Hull, T. E., Keenan, T. A., Kehl, W. B., McCluskey, E. J., Navarro, S. O., Rheinboldt, W. C., Schweppe, E. J., Viavant, W., & Young, D. M., Jr. (1968). Curriculum 68: Recommendations for academic programs in computer science: a report of the ACM curriculum committee on computer science. *Commun. ACM, 11*(3), 151-197.

Bellamy, R. (1994). What does pseudo-code do? A psychological analysis pf the use of pseudo-code by experienced programmers. *Human-Computer Interaction, 9*, 225-246.

Bennedsen, J., & Caspersen, M. E. (2005). *An investigation of potential success factors for an introductory model-driven programming course.* Proceedings of the 2005 international workshop on Computing education research Seattle, WA, USA.

Bennedsen, J., & Caspersen, M. E. (2006). Abstraction ability as an indicator of success for learning object-oriented programming? *SIGCSE Bull., 38*(2), 39-43.

Benson, B., & Hartz, M. (2000). A comparison of observational studies and randomized controlled trials. *The New England Journal of Medicine, 342*, 1898-1886.

Bergadano, F., Gunetti, D., & Picardi, C. (2002). User authentication through keystroke dynamics. *ACM Trans. Inf. Syst. Secur., 5*(4), 367-397.

Bergin, S., & Reilly, R. (2005). *Programming: factors that influence success*. Proceedings of the 36th SIGCSE technical symposium on Computer science education, St. Louis, Missouri, USA.

Bergland, A., & Wiggberg, M. (2006). *Students learn CS in different ways: Insights from an empirical study.* Proceedings of the ITiCSE'06, Bologna, Italy.

Bhargava, H. K. (1999). Data mining by decomposition: Adaptive search for hypothesis generation. *INFORMS J. on Computing, 11*(3), 239-247.

Biggs, J. B., & Moore, P. (1993). *The Process of Learning*. Sydney: Prentice Hall.

Biggs, J. B., & Tang, C. (2007). *Teaching for quality learning at university : what the student does* (3rd ed.). Maidenhead: McGraw-Hill/Society for Research into Higher Education & Open University Press.

Bishop-Clark, C., & Wheeler, D. D. (1994). The Myers-Briggs personality type and its relationship to computer programming. *Journal of Research on Computing in Education, 26*(3), 358-370.

Black, A. P., Bruce, K. B., Homer, M., Noble, J., Ruskin, A., & Yannow, R. (2013). *Seeking grace: a new object-oriented language for novices*. Proceedings of the Proceeding of the 44th ACM technical symposium on Computer science education, Denver, Colorado, USA.

Blackwell, A. F., Whitley, K. N., Good, J., & Petre, M. (2001). Cognitive factors in programming with diagrams. *Artificial Intelligence Review, 15*(1-2), 95-114.

Bloom, B. (1984). *Taxonomy of educational objectives*. Boston, MA: Allyn and Bacon.

Bonham, L. A. (1998). Learning Style Instruments: Let the buyer beware. *Lifelong Learning, 11*(6), 12-16.

Brooks, R. (1983). Towards a theory of the cognitive processes in computer programming. *International Journal of Man-Machine Studies, 9*, 543-554.

Byrne, P., & Lyons, G. (2001). *The effect of student attributes on success in programming*. Proceedings of the 6th annual conference on Innovation and technology in computer science education, Canterbury, United Kingdom.

Cantwell Wilson, B., & Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. *Technical Symposium on Computer Science Education*, 184-188.

Carbone, A., Hurst, J., Mitchell, I., & Gunstone, D. (2009). *An exploration of internal factors influencing student learning of programming*. Proceedings of the Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95, Wellington, New Zealand.

Cilliers, C., Calitz, A., & Greyling, J. (2005). The effect of integrating an Iconic programming notation into CS1. *SIGCSE Bull., 37*(3), 108-112. doi: 10.1145/1151954.1067478

Clark, D. (1999, 2/4/2000). The Hawthorne Effect Retrieved 15/4/2008, 2008, from http://www.nwlink.com/~Donclark/hrd/history/hawthorne.html

Coffield, F., Moseley, D., Hall, E., & Ecclestone, K. (2004). Learning styles and pedagogy in post-16 learning: A systematic and critical review. Trowbridge, Wiltshire: Learning and Skills Research Centre.

Comer, D. E., Gries, D., Mulder, M., Tucker, A., Turner, A. J., & Young, P. R. (1989). Computing as a discipline. *Commumications of the ACM, 32*(1), 9-23.

Cook, D. (1967). The impact of the Hawthorne Effect in experimental designs in eductional research. Columbus: Ohio State University.

Cook, J. E., & Wolf, A. L. (1995). Automating process discovery through event-data analysis. *ACM Transactions on Software Engineering and Methodology, 7*(3), 73-82.

Cross, J. H., & Barowski, L. A. (2002). The jGrasp Handbook: School of Engineering, Auburn University.

Cross, J. H., Hendrix, T. D., Jain, J., & Barowski, L. A. (2007). Dynamic object viewers for data structures. *ACM SIGCSE Bulletin, 39*(1), 4-8.

Cross, J. H., & Sheppard, S. V. (1988). *Graphical Extensions For Pseudo-Code, PDLs, and Source Code.* Proceedings of the ACM 16th Annual Conference on Computer Sceince, Atlanta.

Dale, N. (2002). Increasing interest in CS ed research. *SIGCSE Bull., 34*(4), 16-17.

de Raadt, M., Toleman, M., & Watson, H. J. (2004). Training strategic problem solvers. *SIGCSE Bull., 36*(2), 48-51. doi: 10.1145/1024338.1024370

Denny, P., Luxton-Reilly, A., & Tempero, E. (2012). *All syntax errors are not equal*. Proceedings of the Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education, Haifa, Israel.

DePasquale, P., Lee, J., & Perez-Quizones, D. (2004). Evaluation of subsetting programming language elements in a novice's programming environment. *SIGCSE Bull., 36*(1), 260-264. doi: http://doi.acm.org/10.1145/1028174.971392

Dingle, A., & Zander, C. (2000). *Assessing the ripple effect of CS1 language choice.* Proceedings of the Second annual Consortium for Computing Sciences in Colleges, Beaverton, Oregon, US.

Dourish, P. (2001). *Where the action is: The foundations of embodied interaction*. Cambridge, MA: MIT Press.

Drysdale, M., Ross, J., & Schulz, R. (2001). Cognitive Learning styles and academic performance in 19 first-year university courses: Successful students versus students at risk. *Journal of Education for Students Placed at Risk, 6*(3), 271-289.

Dunn, R., Beaudry, J., & Klavas, A. (1989). Survey of research on Learning styles. *Educational Leadership, 46*(6), 50-58.

Eckerdal, A., & Berglund, A. (2005). *What does it take to learn 'Programming Thinking'?* Proceedings of the 2005 international workshop on Computer Science Education Research, Seattle, WA, USA.

Edwards, S. H., Snyder, J., Perez-Quinones, M. A., Allevato, A., Kim, D., & Tretola, B. (2009). *Comparing effective and ineffective behaviors of student programmers*. Proceedings of the Fifth international workshop on Computing education research workshop, Berkeley, CA, USA.

Felder, R. (1993). Reaching the second tier: Learning and teaching styles in college science education. *Journal of College Science Teaching, 23*(5), 286 - 290.

Felleisen, M., Findler, R., Flatt, M., & Krishnamurthi, S. (1998). The DrScheme project: an overview. *ACM SIGPLAN Notices, 33*(6), 17-23.

Fenwick, J. B., Jr, Norris, C., Barry, F. E., Rountree, J., Spicer, C. J., & Cheek, S. D. (2009). Another look at the behaviors of novice programmers. *SIGCSE Bull., 41*(1), 296-300.

Fincher, S., & Petre, M. (2004). *Computer Science Education Research*. London: RoutledgeFalmer.

Fisher, C., & Sanderson, P. (1996). Exploratory sequential data analysis: Exploring continuous observational data. *Interactions, 3*(2), 25-34.

Fosnot, C. (1996). Constructivism:theory, perspectives and practice. *Teachers College Press, New York*.

Fowler, L., Campbell, V., McGill, D., & Roy, G. G. (2003). *An innovative approach to teaching first year programming supported by learning style investigation.* Proceedings of the 14th Annual Conference for Australasian Association for Engineering Education, Melbourne, Australia.

Fraenkel, J., & Wallen, N. (2006). *How to design and evaluate research in education*. Boston: McGraw-Hill.

Freund, S., & Roberts, E. (1996). Thetis: an ANSI C programming environment designed for introductory use. *ACM SIGCSE Bulletin, 28*(1), 300-304. doi: http://doi.acm.org/10.1145/236462.236560

Fuller, U., Johnson, C., Ahoniemi, T., Hernan-Losada, I., Jackova, J., Lahtinen, E., Lewis, T., McGee Thompson, D., Riedesel, C., & Thompson, E. (2007). Developing a computer science-specific learning taxonomy. *ACM SIGCSE Bulletin, 39*(4), 152-170.

Galliers, R. D. (1990). *Choosing appropriate Information Systems research approaches: A revised taxonomy.* Proceedings of the Information Systems Research Arena of the 90s: Challenges, Perceptions and Alternative Approaches, Copenhagen, Denmark.

Geiger, M., & Pinto, J. (1991). Changes in learning style preference during a three-year longitudinal study. *Psychological Reports, 69*, 755-762.

Gilmore, D. J. (1990). Methodological issues in the study of programming. In J.-M. Hoc, T. R. G. Green, R. Samurcay & D. J. Gilmore (Eds.), *Psychology of Programming* (pp. 83-98). London: Harcourt Brace Jovanovich Publishers.

Glaser, B. G., & Strauss, A. L. (1967). *Discovery of Grounded Theory: Strategies for Qualitative Research*. New Brunswick, U.S.A: Aldine Transaction.

Goldstein, H., & von Neumann, J. (1947). Planning and coding problems for an electronic computing instrument. part II, vol1. *prepared for the U.S. Army Ordinance Dept.*

Gomes, A., & Mendes, A. (2010). *A study on student performance in first year CS courses*. Proceedings of the 2010 ACM SIGCSE Annual Conference on Innovation and Technology in Computer Science Education.

Gomes, A., Santos, Á. N., & Mendes, A. J. (2012). *A study on students' behaviours and attitudes towards learning to program*. Proceedings of the Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education, Haifa, Israel.

Goold, A., & Rimmer, R. (2000). Indicators of performance in first-year computing. *ACM SIGCSE Bulletin, 32*(2).

Gray, P., McLeod, I., Draper, S., Crease, M., & Thomas, R. (2004). *A distributed usage monitoring system*. Proceedings of the CADUI, Funchal, Portugal.

Gregorc, A. (1979). Learning/teaching styles: potent forces behind them. *Educational Leadership, 36*, 234-237.

Greyling, J. H., Cilliers, C. B., & Calitz, A. P. (2006, 10-13 July 2006). *B#: The Development and Assessment of an Iconic Programming Tool for Novice Programmers.* Proceedings of the Information Technology Based Higher Education and Training, 2006. ITHET '06. 7th International Conference on.

Gross, P., & Powers, K. (2005). *Evaluating assessments of novice programming environments.* Proceedings of the Proceedings of the First International Workshop on Computing Education Research.

Gugerty, L., & Olson, G. M. (1986, 1986). *Debugging by skilled and novice programmers.* Proceedings of the CHI'86: Human Factors in Computing Systems, New York.

Guo, P. J. (2013). *Online python tutor: embeddable web-based program visualization for cs education*. Proceedings of the Proceeding of the 44th ACM technical symposium on Computer science education, Denver, Colorado, USA.

Guzdial, M. (1993). Deriving software usage patterns from log files: Graphics, Visualization, and Usability Center, Georgia Institute of Technology.

Guzdial, M. (2004). Programming Environments for Novices. In S. Fincher & M. Petre (Eds.), *Computer Science Education Research* (pp. 127-154). Lisse, The Netherlands: Taylor & Francis.

Hagan, D., & Markham, S. (2000). Does it help to have some programming experience before beginning a computing degree program? *SIGCSE Bull., 32*(3), 25-28.

Harasym, P., Leong, E., Lucier, G., & Lorscheider, F. (1995). Gregorc learning styles and achievement in anatomy and physiology. *American Journal of Physiology, 268*(6 pt 3), 56-60.

Hilbert, D. M., & Redmiles, D. F. (2000). Extracting usability information from user interface events. *ACM Computing Surveys, 32*(4).

Hoaglin, D. C., Mosteller, F., & Tukey, J. W. (1983). *Understanding robust and exploratory data analysis*. New York: John Wiley & Sons, Inc.

Hoc, J.-M. (1990). *Psychology of programming*. London: Academic.

Humphrey, W. (1995). *A discipline for Software Engineering*. Boston, Mass.: Addison-Wesley.

Hundhausen, C., Brown, J., Farley, S., & Skarpas, D. (2006). *A methodology for analyzing the temporal evolution of novice programs based on semantic components.* Proceedings of the 2006 international workshop on Computing education research Canterbury, United Kingdom

Isohanni, E., & Knobelsdorf, M. (2010). *Behind the curtain: students' use of VIP after class*. Proceedings of the Proceedings of the Sixth international workshop on Computing education research, Aarhus, Denmark.

Jadud, M. (2005a). A first look at novice compilation behavior. *Computer Science Education, 15*(1), 25-40.

Jadud, M. (2005b). A first look at novice compilation behaviour. *Computer Science Education, 15*(1), 25-40.

Jadud, M. (2006). *An exploration of novice compilation behaviour in BlueJ.* PhD, University of Kent, Canterbury.

Jenkins, T., & Davy, J. (2002). Diversity and motivation in introductory programming. *Innovations in Teaching And Learning in Information and Computer Sciences*.

Johnson, P. M., Kou, H., Agustin, J., Chan, C., Moore, C., Miglani, J., Zhen, S., & Doane, W. E. J. (2003). *Beyond the Personal Software Process: metrics collection and analysis for the differently disciplined*. Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon.

Jonassen, D. H., & Grabowski, B. L. (1993). *Handbook of individual differences learning and instruction*. London: Lawrence Erlbaum Associates.

Jones, S., & Burnett, G. (2007). *Spatial ability and learning to program.* Proceedings of the Psychology of Programming Interest Group '07.

Joy, M., Sinclair, J., Sun, S., Sitthiworachart, J., & López-González, J. (2009). Categorising computer science education research. *Education and Information Technologies, 14*(2), 105-126.

Judd, T. S., & Kennedy, G. (2004). *More sense from audit trails: Exploratory sequential data analysis.* Proceedings of the ASCILITE, Perth, Western Australia.

Kaneiwa, K., & Tojo, S. (2005). *Logical aspects of events: Quantification, sorts, composition and disjointness*. Proceedings of the Australasian Ontology Workshop, Conference in Research and Practice in Information Technology, Sydney, Australia.

Ko, A., Aung, H., & Myers, B. (2005). *Eliciting design requirements for maintenance-oriented IDEs: a detailed study of corrective and*

perfective maintenance tasks. Proceedings of the International Conference on Software Engineering, St. Louis, MO.

Kolb, D. A. (1981). Experiential learning theory and the Learning Style Inventory: A reply to Freedman and Stumpf. *The Academy of Management Review, 6*, 289-296.

Kolb, D. A. (1984). *Experiential learning experience as the source of learning and development*. Englewood Cliffs, NJ: Prentice-Hall.

Kolb, D. A. (2000). *Facilitator's guide to learning*. Boston: Hay Group Transforming Learning.

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ system and its pedagogy. *Journal of Computer Science Education, Special Issue on Learning and Teaching Object Technology, 13*(4).

Kölling, M., & Rosenberg, J. (1996a). Blue - A Language for teaching object-oriented programming. *ACM SIGCSE Bulletin, 28*(1).

Kölling, M., & Rosenberg, J. (1996b). An object-oriented program development environment for the first programming course. *ACM SIGCSE Bulletin, 28*(1).

Kölling, M., & Rosenberg, J. (2001). Guidelines for teaching object orientation with Java. *ACM SIGCSE Bulletin, Proceedings of the 6th annual conference on Innovation and technology in compuer science education., 33*(3), 33-36.

Kölling, M., & Rosenberg, J. (2002). BlueJ - the hitch-hikers guide to object orientation (pp. 8): The Maersk Mc-Kinney Moller Institute for Production Technology, University of Southern Denmark.

Lemos, R. (1979). Teaching programming languages: A survey of approaches. *Proceedings of the tenth SIGCSE technical symposium on Computer science education* 174-181

Lister, R. (2010). CS EDUCATION RESEARCH: The naughties in CSEd research: a retrospective. *ACM Inroads, 1*(1), 22-24.

Lister, R., Adams, E., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Mostrom, J., Sanders, K., Seppala, O., Simon, B., & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin, 36*(4), 119-150.

Lister, R., Clear, T., Simon, Bouvier, D. J., Carter, P., Eckerdal, A., Jackov, J., Lopez, M., McCartney, R., Robbins, P., Sepp, O., & Thompson, E. (2010). Naturally occurring data as research instrument: analyzing examination responses to study the novice programmer. *SIGCSE Bull., 41*(4), 156-173.

Loo, R. (1999). Confirmatory factor analysis of Kolb's Learning Style Inventory (LSI-1985). *British Journal of Educational Psychology, 69*(2), 213-219.

Mainemelis, C., Boyatzis, R. E., & Kolb, D. A. (2002). Learning styles and adaptive flexibility: testing experiential learning theory. *Management Learning, 33*(1), 5-33.

Mayer, R. E. (1981). The psychology of how novices learn computer programming. *Computer Surveys, 13*(1), 121-140.

McCracken, M., Wilusz, T., Alstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., & Utting, I. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin, 33*(4), 125-180.

McDonald, J. H. (2009). *Handbook of Biological Statistics* (2nd ed.). Baltimore, Maryland: Sparky House Publishing.

McGettrick, A., Boyle, R., Ibbett, R., Lloyd, J., Lovegrove, G., & Mander, K. (2005). Grand challenges in computing: Education--a summary. *The Computer Journal, 48*, 42-48.

McKeachie, W. J. (1995). Learning styles can become learning strategies. *The National Teaching and Learning Forum, 4*(6), 1-3.

Mierle, K., Laven, K., Roweis, S., & Wilson, G. (2005). Mining student CVS repositories for performance indicators. *SIGSOFT Softw. Eng. Notes, 30*(4), 1-5.

Murphy, C., Kaiser, G., Loveland, K., & Hasan, S. (2009). Retina: helping students and instructors based on observed programming activities. *SIGCSE Bull., 41*(1), 178-182.

Norris, C., Barry, F., Fenwick, J. B., Jr., Reid, K., & Rountree, J. (2008). ClockIt: collecting quantitative data on how beginning software developers really work. *SIGCSE Bull., 40*(3), 37-41.

Paschler, H., McDaniel, M., Rohrer, D., & Bjork, R. (2008). Learning styles: Concepts and evidence. *Psychological Science in the Public Interest, 9*(3), 105-119.

Pattis, R. E. (1981). *Karel the Robot*. New York: Wiley.

Pears, A., Seidman, S., Eney, C., Kinnumem, P., & Malmi, L. (2005). Constructing a core literature for computing education research. *ACM SIGCSE Bulletin, 37*(4), 152-161.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. [Working Group report]. *ACM SIGCSE Bulletin, 39*(4), 204-223.

Pennington, N., & Grabowski, B. L. (1990). The tasks of programming. In J.-M. Hoc (Ed.), *Psychology of programming* (pp. 45-61). London: Harcourt Brace Jovanovich.

Perkins, D., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1989). Conditions of learning in novice programmers. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 261-279).

Petre, M. (1990). Expert Programmers and Programming Languages. In J.-M. Hoc, T. R. G. Green, R. Samurcay & D. J. Gilmore (Eds.), *Psychology of programming* (pp. 103-115). London: Harcourt Brace Jovanovich.

Pillay, N., & Jugoo, V. (2005). An investigation into student characteristics affecting novice programming performance. *SIGCSE Bull., 37*(4), 107-110.

Ping, C., & Garcia, W. (2010, 7-9 July 2010). *Hypothesis generation and data quality assessment through association mining.* Proceedings of the Cognitive Informatics (ICCI), 2010 9th IEEE International Conference on, Beijing, China.

Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. *Proceedings of the 9th annual SIGCSE Conference on innovation and technology in computer science education*, 171-175.

Randolph, J. (2007). *Computer Science Education Research at the the crossroads: A methodological review of Computer Science Education Research: 2000 - 2005.* Doctor of Philosophy, Utah State University, Logan, Utah.

Renaud, K., & Gray, P. (2004). *Making sense of low-level usage data to understand user activities*. Proceedings of the SAICSIT.

Roberts, E., Engel, G., Chang, C., Cross, J., Shackelford, R., Sloan, R., Carver, D., Eckhouse, R., King, W., Lau, F., Srimani, P., Austing, R., Cover, C., Davies, G., McGettrick, A., Schneider, G., & Wolz, U. (2001). Computing Curricula 2001: Computer Science. Retrieved from www.acm.org/sigcse/cc2001/cc2001.pdf

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137-172.

Rountree, N., Rountree, J., Robins, A., & Hannah, R. (2004). Interacting factors that predict success and failure in a CS1 course. *SIGCSE Bull., 36*(4), 101-104. doi: 10.1145/1041624.1041669

Roy, G. (2006). Designing and explaining programs with a literate pseudocode. *Journal on Educational Resources in Computing (JERIC), 6*(1).

Sanjaniemi, J., & Kuittinen, M. (2005). An experiment on using roles of variables in teaching introductory programming. *Computer Science Education, 15*(1), 59-82.

Sankoff, D., & Kruskal, J. (1983). *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison*. Reading, Mass.: Addison-Wesley Pub,.

Searson, R., & Dunn, R. (2001). The learning-style teaching model. *Science and Children, 38*(5), 22 - 26.

Shackelford, R., McGettrick, A., Sloan, R., Topi, H., Davies, G., Kamali, R., Cross, J., Impagliazzo, J., LeBlanc, R., & Lunt, B. (2006). Computing Curricula 2005: The Overview Report. *SIGCSE Bull., 38*(1), 456-457. doi: 10.1145/1124706.1121482

Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E., & Whalley, J. L. (2008). *Going SOLO to assess novice programmers*. Proceedings

of the Proceedings of the 13th annual conference on Innovation and technology in computer science education, Madrid, Spain.

Shneiderman, B., Mayer, R. E., McKay, D., & Heller, P. (1977). Experimental investigations of the utility of detailed flowcharts in programming. *Commumications of the ACM, 20*(6), 373-381.

Simon. (2007). A classification of recent Australasian computing education publications. *Computer Science Education, 17*(3), 155-169.

Simon, Carbone, A., de Raadt, M., Lister, R., Hamilton, M., & Sheard, J. (2008). *Classifying computing education papers: Process and results*. Proceedings of the Fourth international Workshop on Computing Education Research, Sydney, Australia.

Simon, Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., Raadt, M. d., Haden, P., Hamer, J., Hamilton, M., Lister, R., Petre, M., Sutton, K., Tolhurst, D., & Tutty, J. (2006). *Predictors of success in a first programming course*. Proceedings of the Proceedings of the 8th Australasian Conference on Computing Education - Volume 52, Hobart, Australia.

Smith, W., Sekar, S., & Townsend, K. (2002). *The impact of surface and relective teaching and learning on student academic success.* Proceedings of the 7th Annual European Learning Styles Information Network Conference, Ghent.

Soloman, B., & Felder, R. (1999). Index of Learning Styles (ILS), Retrieved 7/9/2012, 2012, from http://www4.ncsu.edu/unity/lockers/users/f/felder/public/ILSpage

Soloway, E. (1986). Learning to program = Learning to construct mechanisms and explanations. *Communications of the ACM, 29*(9), 850-858.

Spacco, J., Strecker, J., Hovemeyer, D., & Pugh, W. (2005). Software repository mining with Marmoset: An automated programming project snapshot and testing system. *SIGSOFT Softw. Eng. Notes, 30*(4), 1-5.

Tan, P.-N., Steinback, M., & Kumar, V. (2006). *Introduction to data mining*. Boston: Pearson.

Tenenberg, J., Impagliazzo, J., Fincher, S., & Joyce, D. (2008). *Publishing in computing education.* Proceedings of the SIGCSE'08, Portland, Oregon, USA.

Thomas, L., Ratcliffe, M., Woodbury, J., & Jarman, E. (2002). Learning styles and performance in the introductory programming sequence. *SIGCSE Bull., 34*(1), 33-37.

Valentine, D. (2004). *CS educational research: A meta-analysis of SIGCSE Technical Symposium Proceedings.* Proceedings of the 35th SIGCSE technical symposium on Computer science education . Norfolk, VA, USA.

Venables, A., Tan, G., & Lister, R. (2009). *A closer look at tracing, explaining and code writing skills in the novice programmer*.

Proceedings of the Fifth international workshop on Computing education research workshop, Berkeley, CA, USA.

Ventura, P. (2003). *On the origins of programmers: Identifying predictors of success for an objects first CS1*. PhD dissertation. The State University of New York at Buffalo.

Ventura, P., & Ramamurthy, B. (2004). *Wanted: CS1 students. No experience required*. Proceedings of the SIGCSE '04, Norfolk, Virginia, USA.

Vortac, O. U., Edwards, M. B., & Manning, C. A. (1994). Sequences of actions for individual and teams of air traffic controllers. *Human-Computer Interaction, 9*, 319-343.

Weber Becker, B. (2001). Teaching CS1 with Karel the Robot in Java. *ACM SIGCSE Bulletin, 33*(1), 50-54.

Weinberg, G. M. (1971). *The psychology of computer programming*. New York: Van Nostrand Reinhold.

Whipkey, K. (1984). Identifying predictors of programming skill. *ACM SIGCSE Bulletin, 16*(4), 36-42.

Wiedenbeck, S. (2005). *Factors affecting the success of non-majors in learning to program*. Proceedings of the 2005 international workshop on Computing education research, Seattle, WA, USA.

Winslow, L. E. (1996). Programming pedagogy -- A psychological overview. *ACM SIGCSE Bulletin, 28*(3), 17-22.

Wirth, N. (1976). *Algorithm + Data Structures = Programs*: Prentice Hall.